Machine Learning: The Basics

Alexander Jung, August 27, 2023

please cite as:

| A. Jung, "Machine Learning: The Basics," Springer, Singapore, 2022

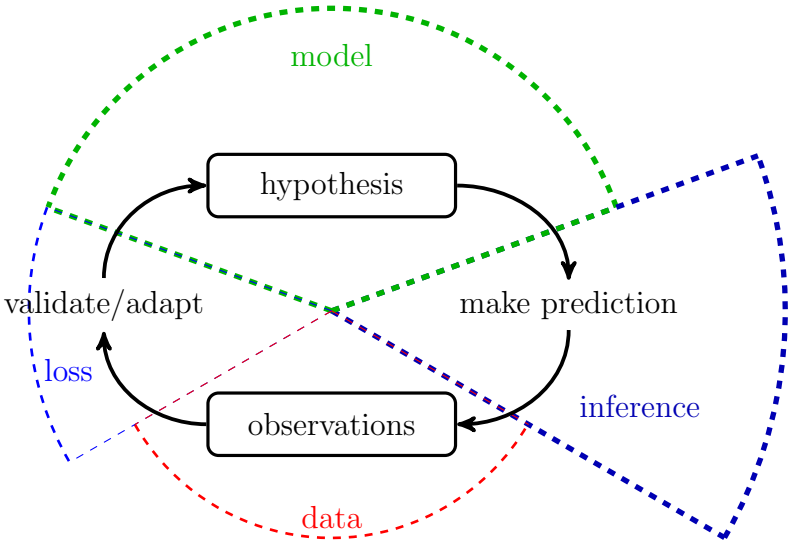


Figure 1: Machine learning combines three main components: model, data and loss. Machine learning methods implement the scientific principle of "trial and error". These methods continuously validate and refine a model based on the loss incurred by its predictions about a phenomenon that generates data.

Preface

Machine learning (ML) influences our daily lives in several aspects. We routinely ask ML empowered smartphones to suggest lovely restaurants or to guide us through a strange place. ML methods have also become standard tools in many fields of science and engineering. ML applications transform human lives at unprecedented pace and scale.

This book portrays ML as the combination of three basic components: data, model and loss. ML methods combine these three components within computationally efficient implementations of the basic scientific principle "trial and error". This principle consists of the continuous adaptation of a hypothesis about a phenomenon that generates data.

ML methods use a hypothesis map to compute predictions of a quantity of interest (or higher level fact) that is referred to as the label of a data point. A hypothesis map reads in low level properties (referred to as features) of a data point and delivers the prediction for the label of that data point. ML methods choose or learn a hypothesis map from a (typically very) large set of candidate maps. We refer to this set as of candidate maps as the hypothesis space or model underlying an ML method.

The adaptation or improvement of the hypothesis is based on the discrepancy between predictions and observed data. ML methods use a loss function to quantify this discrepancy. A plethora of different ML methods is obtained by different design choices for the data (representation), model and loss. ML methods also differ vastly in their practical implementations which might obscure their unifying basic principles. Two examples for such ML methods are deep learning and linear regression.

Deep learning methods use cloud computing frameworks to train large models on large datasets. Operating on a much finer granularity for data and computation, linear regression can typically be implemented on small embedded systems. Nevertheless, deep learning methods and linear regression use the very same principle of iteratively updating a model based on the discrepancy between model predictions and actual observed data. This discrepancy is measured via a loss function.

We believe that thinking about ML as combinations of three components given by data, model and loss function helps to navigate the steadily growing offer for ready-to-use ML methods. Our three-component picture allows a unified treatment of ML techniques, such as "early stopping", "privacy-preserving" ML and "explainable ML", that might seem unrelated at first sight. The regularization effect of early stopping in gradient-based methods arises from the shrinking of the effective hypothesis space. Privacy-preserving ML methods can be obtained by particular choices for the features used to characterize data points (see Section 9.5). Explainable ML methods can be obtained by particular choices for the hypothesis space and loss function (see Chapter 10).

To make good use of ML tools it is instrumental to understand its underlying principles at the appropriate level of detail. It is typically not necessary to understand the mathematical details of advanced optimization methods to successfully apply deep learning methods. On a lower level, this tutorial helps ML engineers choose suitable methods for the application at hand. The book also offers a higher level view on the implementation of ML methods which is typically required to manage a team of ML engineers and data scientists.

Acknowledgement

This book grew from lecture notes prepared for - and the student feedback received from - the courses CS-E3210 "Machine Learning: Basic Principles", CS-E4800 "Artificial Intelligence", CS-EJ3211 "Machine Learning with Python", CS-EJ3311 "Deep Learning with Python" and CS-C3240 "Machine Learning" offered at Aalto University and within the Finnish University network `fitech.io`. The author is indebted to Shamsiiat Abdurakhmanova, Tomi Janhunen, Yu Tian, Natalia Vesselinova, Linli Zhang, Ekaterina Voskoboinik, Buse Atli, Stefan Mojsilovic for carefully reviewing early drafts of this book. Some of the figures have been generated with the help of Linli Zhang. The author is also grateful for the feedback received from Luca Vassio, Lasse Leskelä, Jukka Suomela, Ekkehard Schnoor, Homayun Afrabandpey, Linli Zhang, Väinö Mehtola, Anselmi Jokinen, Oleg Vlasovetc, Anni Niskanen, Dominik Hodan, Georgios Karakasidis, Yuvrajsinh Chudasama, Joni Pääkkö, Harri Wallenius, Nuutti Sten, ST John, Antti Ainamo, Arastoo Ajorian, Sorin Patrasoiu and Satu Korhonen.

Lists of Symbols

Sets

$a \in \mathcal{A}$	This statement indicates that the object a is an element of the set $\mathcal{A}$.
$a := b$	This statement defines a to be shorthand for b.
$\lvert\mathcal{A}\rvert$	The cardinality (number of elements) of a finite set $\mathcal{A}$.
$\mathcal{A} \subseteq \mathcal{B}$	$\mathcal{A}$ is a subset of $\mathcal{B}$.
$\mathcal{A} \subset \mathcal{B}$	$\mathcal{A}$ is a strict subset of $\mathcal{B}$.
$\mathbb{N}$	The set of natural numbers $1, 2, \ldots$.
$\mathbb{R}$	The set of real numbers x [120].
$\mathbb{R}_+$	The set of non-negative real numbers $x \geq 0$.
$\{0, 1\}$	The set consisting of two real-number 0 and 1.
$[0, 1]$	The closed interval of real numbers x with $0 \leq x \leq 1$.
$f(\cdot), h(\cdot)$	A function or map $f(\cdot)$ that accepts any element $a \in \mathcal{A}$ from a set $\mathcal{A}$ as input and delivers a well-defined element $f(a) \in \mathcal{B}$ of a set $\mathcal{B}$. The set $\mathcal{A}$ is the domain of the function f and the set $\mathcal{B}$ is the codomain of f. Machine Learning revolves around finding (or learning) a function h (which we call hypothesis) that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $h(\mathbf{x})$ for the label y of the data point.

Matrices and Vectors

$\mathbf{I}_{l\times n}$	A generalized identity matrix with l rows and n columns. The entries of $\mathbf{I}_{l\times n} \in \mathbb{R}^{l\times n}$ are equal to 1 along the main diagonal and equal to 0 otherwise. For example, $\mathbf{I}_{1\times 2} = \big(1, 0\big)$ and $\mathbf{I}_{2\times 1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.
$\mathbf{I}$	A square identity matrix whose shape should be clear from the context.
$\mathbb{R}^n$	The set of vectors $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T$ consisting of n real-valued entries $x_1, \ldots, x_n \in \mathbb{R}$.
$\mathbf{x} = \big(x_1, \ldots, x_n\big)^T$	A vector of length n. The jth entry of the vector is denoted x_j.
$\|\mathbf{x}\|_2$	The Euclidean (or "ℓ_2") norm of the vector $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$ given as $\|\mathbf{x}\|_2 := \sqrt{\sum_{j=1}^n x_j^2}$.
$\|\mathbf{x}\|$	Some norm of the vector $\mathbf{x} \in \mathbb{R}^n$ [46]. Unless specified otherwise, we mean the Euclidean norm $\|\mathbf{x}\|_2$.
$\mathbf{x}^T$	The transpose of a vector $\mathbf{x}$ that is considered a single column matrix. The transpose is a single-row matrix $\big(x_1, \ldots, x_n\big)$.
$\mathbf{X}^T$	The transpose of a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$. A square real-valued matrix $\mathbf{X} \in \mathbb{R}^{m\times m}$ is called symmetric if $\mathbf{X} = \mathbf{X}^T$.
$\mathbf{0} = \big(0, \ldots, 0\big)^T$	A vector of zero entries whose length should be clear from context.
$\big(\mathbf{v}^T, \mathbf{w}^T\big)^T$	The vector of length $n+n'$ obtained by concatenating the entries of vector $\mathbf{v} \in \mathbb{R}^n$ with the entries of $\mathbf{w} \in \mathbb{R}^{n'}$.
$\mathrm{span}\{\mathbf{B}\}$	The span of a matrix $\mathbf{B} \in \mathbb{R}^{a\times b}$, which is the subspace of all linear combinations of columns of $\mathbf{B}$, $\mathrm{span}\{\mathbf{B}\} = \big\{\mathbf{Ba} : \mathbf{a} \in \mathbb{R}^b\big\} \subseteq \mathbb{R}^a$.
$\mathbb{S}_+^n$	The set of all positive semi-definite (psd) matrices of size $n \times n$.

Probability Theory

$\mathbb{E}_p\{f(\mathbf{z})\}$	The expectation of a function $f(\mathbf{z})$ of a RV $\mathbf{z}$ whose probability distribution is $p(\mathbf{z})$. If the probability distribution is clear from context we just write $\mathbb{E}\{f(\mathbf{z})\}$.
$p(\mathbf{x}, y)$	A (joint) probability distribution of a RV whose realizations are data points with features $\mathbf{x}$ and label y.
$p(\mathbf{x}\|y)$	A conditional probability distribution of a RV $\mathbf{x}$ given the value of another RV y [9, Sec. 3.5].
$p(\mathbf{x}; \mathbf{w})$	A parametrized probability distribution of a RV $\mathbf{x}$. The probability distribution depends on a parameter vector $\mathbf{w}$. For example, $p(\mathbf{x}; \mathbf{w})$ could be a multivariate normal distribution of a Gaussian RV $\mathbf{x}$ with the parameter vector $\mathbf{w}$ being the expectation $\mathbb{E}\{\mathbf{x}\}$.
$\mathcal{N}(\mu, \sigma^2)$	The probability distribution of a scalar normal ("Gaussian") RV $x \in \mathbb{R}$ with mean (or expectation) $\mu = \mathbb{E}\{x\}$ and variance $\sigma^2 = \mathbb{E}\big\{(x-\mu)^2\big\}$.
$\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$	The probability distribution of a multivariate (vector-valued) normal ("Gaussian") RV $\mathbf{x} \in \mathbb{R}^n$ with mean (or expectation) $\boldsymbol{\mu} = \mathbb{E}\{\mathbf{x}\}$ and covariance matrix $\mathbf{C} = \mathbb{E}\big\{\big(\mathbf{x}-\boldsymbol{\mu}\big)\big(\mathbf{x}-\boldsymbol{\mu}\big)^T\big\}$.

Machine Learning

$\mathbf{z}$	A data point which is characterized by several properties that we divide into low-level properties (features) and high-level properties (labels) (see Chapter 2).
i	An index $i = 1, 2, \ldots,$ that enumerates the data points within a dataset.
m	The number of data points in (the size of) a dataset.
$\mathcal{D}$	A dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ is a list of individual data points $\mathbf{z}^{(i)}$, for $i = 1, \ldots, m$.
n	The number of individual features used to characterize a data point.
x_j	The j-th feature of a data point. The first feature of a given data point is denoted x_1, the second feature x_2 and so on.
$\mathbf{x}$	The feature vector $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T$ of a data point whose entries are the individual features of the data point. With a slight abuse of language, we refer to the feature vector $\mathbf{x}$ also as "the features" of a data point.
$\mathbf{z}$	Beside the symbol $\mathbf{x}$, we sometimes use $\mathbf{z}$ as another symbol to denote a vector whose entries are features of a data point. We need two different symbols to distinguish between "raw" and learnt feature vectors (see Chapter 9).
$\mathbf{x}^{(i)}$	The feature vector of the ith data point within a dataset.
$x_j^{(i)}$	The jth feature of the ith data point within a dataset.
$\mathcal{B}$	A mini-batch that consists of a (randomly selected) subset of data points from a (typically very large) dataset (see Section 5.7).
B	The size of (the number of data points in) a mini-batch using during a stochastic gradient descent (SGD) step (see Section 5.7).

y	The label (quantity of interest) of a data point.
$y^{(i)}$	The label of the ith data point.
$\big(\mathbf{x}^{(i)}, y^{(i)}\big)$	The features and the label of the ith data point within a dataset.
$\mathcal{X}$	The feature space of a ML method consists of all potential feature values that a data point can have. However, we typically use feature spaces that are much larger than the set of different feature values arising in finite datasets. The majority of the methods discussed in this book uses the feature space $\mathcal{X} = \mathbb{R}^n$ consisting of all Euclidean vectors of length n.
$\mathcal{Y}$	The label space $\mathcal{Y}$ of a ML method consists of all potential label values that a data point can have. We often use label spaces that are larger than the set of different label values arising in a give dataset (e.g., a training set). We refer to a ML problems (methods) using a numeric label space, such as $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = \mathbb{R}^3$, as regression problems (methods). ML problems (methods) that use a discrete label space, such as $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{$"cat", "dog", "mouse"$\}$ are referred to as classification problems (methods).
α	A learning rate or step-size parameter used by gradient-based methods.
$h(\cdot)$	A hypothesis map that reads in features $\mathbf{x}$ of a data point and delivers a prediction $\hat{y} = h(\mathbf{x})$ for its label y.
$\mathcal{H}$	A hypothesis space or model used by a ML method. The hypothesis space consists of different hypothesis maps $h : \mathcal{X} \to \mathcal{Y}$ between which the ML method has to choose.

B^2	The squared bias of a hypothesis $\widehat{h}$ delivered by a ML algorithm that is fed with data points which are modelled as realizations of RVs. Since the data is modelled as realizations of RVs, also the delivered hypothesis $\widehat{h}$ is the realization of a RV.
V	The variance of the (parameters of the) hypothesis delivered by a ML algorithm. If the input data for this algorithm is interpreted as realizations of RVs, so is the delivered hypothesis a realization of a RV.
$L\left((\mathbf{x}, y), h\right)$	The loss incurred by predicting the label y of a data point using the predicted label $\hat{y} = h(\mathbf{x})$. The predicted label $\hat{y}$ is obtained from evaluating the hypothesis $h \in \mathcal{H}$ for the feature vector $\mathbf{x}$ of the data point.
E_v	The validation error of a hypothesis h, which is its average loss incurred over a validation set.
$\widehat{L}\big(h\vert\mathcal{D}\big)$	The empirical risk or average loss incurred by the predictions of hypothesis h for the data points in the dataset $\mathcal{D}$.
E_t	The training error of a hypothesis h, which is its average loss incurred over a training set.
t	A discrete-time index $t = 0, 1, \ldots$ used to enumerate a sequence to sequential events ("time instants").
t	A generic index used to enumerate a finite set of learning tasks within a multi-task learning problem (see Section 7.6).
λ	A regularization parameter that scales the regularization term in the objective function of structural risk minimization (SRM).
$\lambda_j\big(\mathbf{Q}\big)$	The jth eigenvalue (sorted either ascending or descending) of a psd matrix $\mathbf{Q}$. We also use the shorthand λ_j if the corresponding matrix is clear from context.
$\sigma(\cdot)$	The activation function used by an artificial neuron within an artificial neural network (ANN).

$\mathcal{R}_{\hat{y}}$ A hypothesis h partitions the feature space into decision regions. A specific decision region $\mathcal{R}_{\hat{y}}$ consists of all feature vectors $\mathbf{x}$ that are mapped to the same predicted label $h(\mathbf{x}) = \hat{y}$ for all $\mathbf{x} \in \mathcal{R}_{\hat{y}}$.

$\mathbf{w}$ A parameter vector $\mathbf{w} = \big(w_1, \ldots, w_n\big)^T$ whose entries are parameters of a model. These parameters could be feature weights in linear maps, the weights in ANNs or the thresholds used for splits in decision trees.

$h^{(\mathbf{w})}(\cdot)$ A hypothesis map that involves tunable parameters $w_1, \ldots, w_n$ which are stacked into the vector $\mathbf{w} = \big(w_1, \ldots, w_n\big)^T$.

$\nabla f(\mathbf{w})$ The gradient of a differentiable real-valued function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the vector $\nabla f(\mathbf{w}) = \big(\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n}\big)^T \in \mathbb{R}^n$ [119, Ch. 9].

$\phi(\cdot)$ A feature map that reads in features $\mathbf{x} \in \mathcal{X}$ of a data point and delivers new (transformed) features $\widehat{\mathbf{x}} = \phi\big(\mathbf{x}\big) \in \mathcal{X}'$.

Contents

Chapter 1

Introduction

Consider waking up one winter morning in Finland and looking outside the window (see Figure 1.1). It seems to become a nice sunny day which is ideal for a ski trip. To choose the right gear (clothing, wax) it is vital to have some idea for the maximum daytime temperature which is typically reached around early afternoon. If we expect a maximum daytime temperature of around plus 5 degrees, we might not put on the extra warm jacket but rather take only some extra shirt for change.

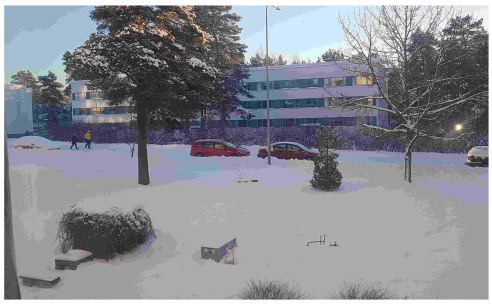


Figure 1.1: Looking outside the window during the morning of a winter day in Finland.

We can use ML to learn a predictor for the maximum daytime temperature for the specific day depicted in Figure 1.1. The prediction shall be based solely on the minimum temperature observed in the morning of that day. ML methods can learn a predictor in a data-driven fashion using historic weather observations provided by the Finnish Meteorological Institute (FMI). We can download the recordings of minimum and maximum daytime temperature for the most recent days and denote the resulting dataset by

$$\mathcal{D} = \big\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big\}. \tag{1.1}$$

Each data point $\mathbf{z}^{(i)} = \big(x^{(i)}, y^{(i)}\big)$, for $i = 1, \ldots, m$, represents some previous day for which the minimum and maximum daytime temperature $x^{(i)}$ and $y^{(i)}$ has been recorded. We depict the data (1.1) in Figure 1.2. Each dot in Figure 1.2 represents a specific day with minimum temperature x and maximum temperature y.

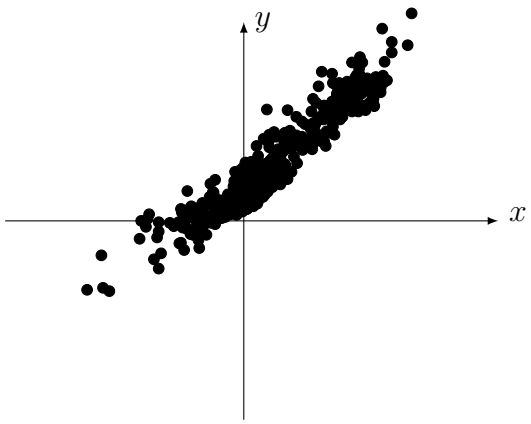


Figure 1.2: Each dot represents a specific day that is characterized by its minimum daytime temperature x as feature and its maximum daytime temperature y as label. These temperatures are measured at some FMI weather station.

ML methods learn a hypothesis $h(x)$, that reads in the minimum temperature x and delivers a prediction (forecast or approximation) $\hat{y} = h(x)$ for the maximum daytime temperature y. Every practical ML method uses a particular hypothesis space out of which the hypothesis h is chosen. This hypothesis space of candidates for the hypothesis map is an important design choice and might be based on domain knowledge.

In what follows, we illustrate how to use domain knowledge to motivate a choice for the hypothesis space. Let us assume that the minimum and maximum daytime temperature of an arbitrary day are approximately related via

$$y \approx w_1 x + w_0 \text{ with some weights } w_1 \in \mathbb{R}_+, w_0 \in \mathbb{R}. \tag{1.2}$$

The assumption (1.2) reflects the intuition (domain knowledge) that the maximum daytime temperature y should be higher for days with a higher minimum daytime temperature x. The assumption (1.2) contains two weights w_1 and w_0. These weights are tuning parameters that allow for some flexibility in our assumption. We require the weight w_1 to be non-negative

but otherwise leave these weights unspecified for the time being. The main subject of this book are ML methods that can be used to learn suitable values for the weights w_1 and w_0 in a data-driven fashion.

Before we detail how ML can be used to find or learn good values for the weights w_0 in w_1 in (1.2) let us interpret them. The weight w_1 in (1.2) can be interpreted as the relative increase in the maximum daytime temperature for an increased minimum daytime temperature. Consider an earlier day with recorded maximum daytime temperature of 10 degrees and minimum daytime temperature of 0 degrees. The assumption (1.2) then means that the maximum daytime temperature for another day with minimum temperature of +1 degrees would be $10 + w_1$ degrees. The second weight w_0 in our assumption (1.2) can be interpreted as the maximum daytime temperature that we anticipate for a day with minimum daytime temperature equal to 0.

Given the assumption (1.2), it seems reasonable to restrict the ML method to only consider linear maps

$$h(x) := w_1 x + w_0 \text{ with some weights } w_1 \in \mathbb{R}_+, w_0 \in \mathbb{R}. \tag{1.3}$$

Since we require $w_1 \geq 0$, the map (1.3) is monotonically increasing with respect to the argument x. Therefore, the prediction $h(x)$ for the maximum daytime temperature becomes higher with higher minimum daytime temperature x.

The expression (1.3) defines a whole ensemble of hypothesis maps. Each individual map corresponding to a particular choice for $w_1 \geq 0$ and w_0. We refer to such an ensemble of potential predictor maps as the model or hypothesis space that is used by a ML method.

We say that the map (1.3) is parametrized by the vector $\mathbf{w} = \big(w_1, w_0\big)^T$ and indicate this by writing $h^{(\mathbf{w})}$. For a given parameter vector $\mathbf{w} = \big(w_1, w_0\big)^T$, we obtain the map $h^{(\mathbf{w})}(x) = w_1 x + w_0$. Figure 1.3 depicts three maps $h^{(\mathbf{w})}$ obtained for three different choices for the parameters $\mathbf{w}$.

ML would be trivial if there is only one single hypothesis. Having only a single hypothesis means that there is no need to try out different hypotheses to find the best one. To enable ML, we need to choose between a whole space of different hypotheses. ML methods are computationally efficient methods to choose (learn) a good hypothesis out of a (typically very large) hypothesis spaces. Note that the hypothesis space constituted by linear maps (1.3) with different weights is already uncountably infinite.

To find, or **learn**, a good hypothesis out of the set (1.3), we need to somehow assess the quality of a particular hypothesis map. ML methods use a loss function for this purpose. A

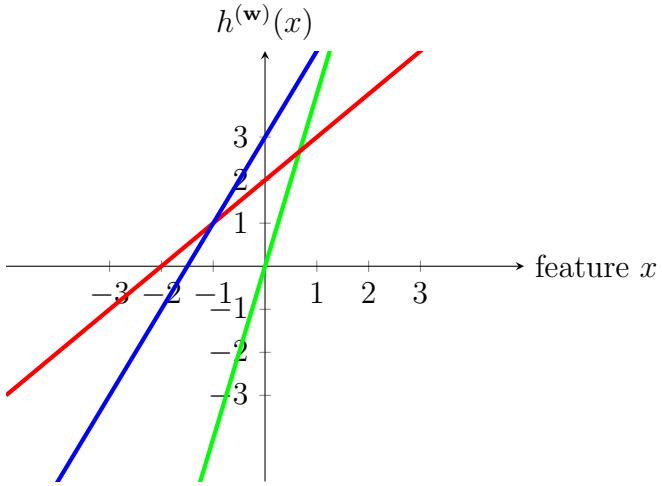


Figure 1.3: Three hypothesis maps of the form (1.3).

loss function is used to quantify the difference between the actual data and the predictions obtained from a hypothesis map (see Figure 1.4). We obtain different ML methods from using different loss functions. One widely-used example of a loss function is the squared error loss $(y - h(x))^2$. The resulting ML method learns a hypothesis from the model (1.3) by tuning w_1, w_0 to minimize the average loss

$$(1/m) \sum_{i=1}^{m} \big(y^{(i)} - h\big(x^{(i)}\big)\big)^2.$$

The above weather prediction is prototypical for many other ML applications. Figure 1 illustrates the typical workflow of a ML method. Starting from some initial guess, ML methods repeatedly improve their current hypothesis based on (new) observed data.

At any given point in time, ML methods use the current hypothesis to predict or (forecast) properties of future observations. The discrepancy between the predictions and the actual observations, as measured using some loss function, is used to improve the current hypothesis. Learning happens by improving the current hypothesis based on the discrepancy between its predictions and the actual observations.

ML methods must start with some initial guess or choice for a good hypothesis. This initial guess can be based on some prior knowledge or domain expertise [96]. While the initial guess for a hypothesis might not be made explicit in some ML methods, each method must use such an initial guess. In our weather prediction application discussed above, we used the linear model (1.2) as the initial hypothesis.

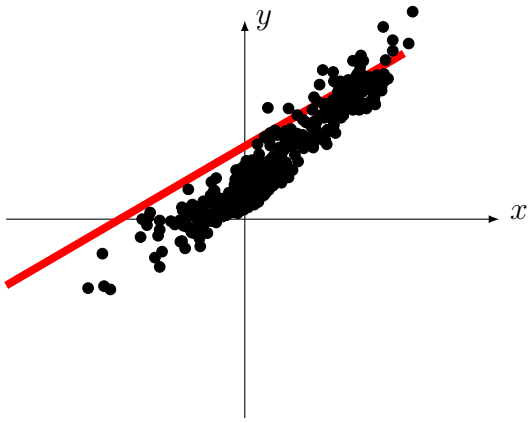


Figure 1.4: Each dot represents a specific days that is characterized by its minimum daytime temperature x and its maximum daytime temperature y. We also depict a straight line representing a linear predictor map. A main principle of ML methods is to learn a predictor (or hypothesis) map with minimum discrepancy between predictor map and data points. Different ML methods use different types of predictor maps (hypothesis space) and loss functions to quantify the discrepancy between hypothesis and actual data points.

1.1 Relation to Other Fields

ML builds on concepts from several other scientific fields. Conversely, ML provides important tools for many other scientific fields.

1.1.1 Linear Algebra

Modern ML methods are computationally efficient methods to fit high-dimensional models to large amounts of data. The models underlying state-of-the-art ML methods can contain billions of tunable or learnable parameters. To make ML methods computationally efficient we need to use suitable representations for data and models.

Maybe the most widely used mathematical structure to represent data is the Euclidean space $\mathbb{R}^n$ with some dimension $n \in \mathbb{N}$ [119]. The rich algebraic and geometric structure of $\mathbb{R}^n$ allows us to design of ML algorithms that can process vast amounts of data to quickly update a model (parameters). Figure 1.5 depicts the Euclidean space $\mathbb{R}^n$ for $n = 2$, which is used to construct scatterplots.

The scatterplot in Figure 1.2 depicts data points (representing individual days) as vectors

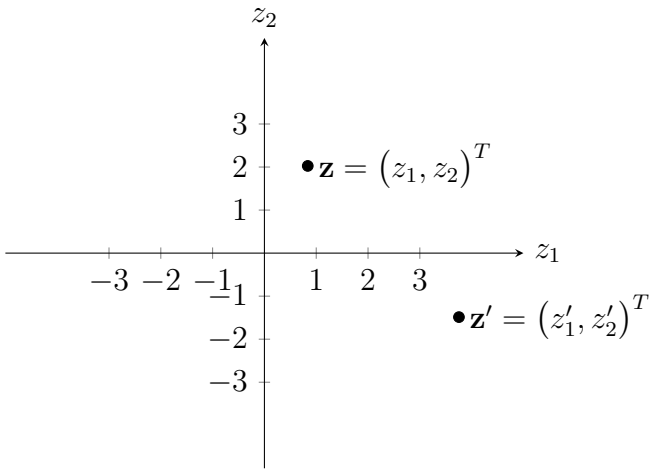


Figure 1.5: The Euclidean space $\mathbb{R}^2$ is constituted by all vectors (or points) $\mathbf{z} = \left(z_1, z_2\right)^T$ (with $z_1, z_2 \in \mathbb{R}$) together with the inner product $\mathbf{z}^T\mathbf{z}' = z_1z_1' + z_2z_2'$.

in the Euclidean space $\mathbb{R}^2$. For a given data point, we obtain its associated vector $\mathbf{z} = (x, y)^T$ in $\mathbb{R}^2$ by stacking the minimum daytime temperature x and the maximum daytime temperature y into the vector $\mathbf{z}$ of length two.

We can use the Euclidean space $\mathbb{R}^n$ not only to represent data points but also to represent models for these data points. One such class of models is obtained by linear maps on $\mathbb{R}^n$. Figure 1.3 depicts some examples for such linear maps. We can then use the geometric structure of $\mathbb{R}^n$, defined by the Euclidean norm, to search for the best model. As an example, we could search for the linear model, represented by a straight line, such that the average (Euclidean) distance to the data points in Figure 1.2 is as small as possible (see Figure 1.4). The properties of linear structures are studied within linear algebra [136]. Some important ML methods, such as linear classifier (see Section 3.1) or PCA (see Section 9.2) are direct applications of methods from linear algebra.

1.1.2 Optimization

A main design principle for ML methods is the formulation of ML problems as optimization problems [133]. The weather prediction problem above can be formulated as the problem of optimizing (minimizing) the prediction error for the maximum daytime temperature. Many ML methods are obtained by straightforward applications of optimization methods to the optimization problem arising from a ML problem (or application).

The statistical and computational properties of such ML methods can be studied using

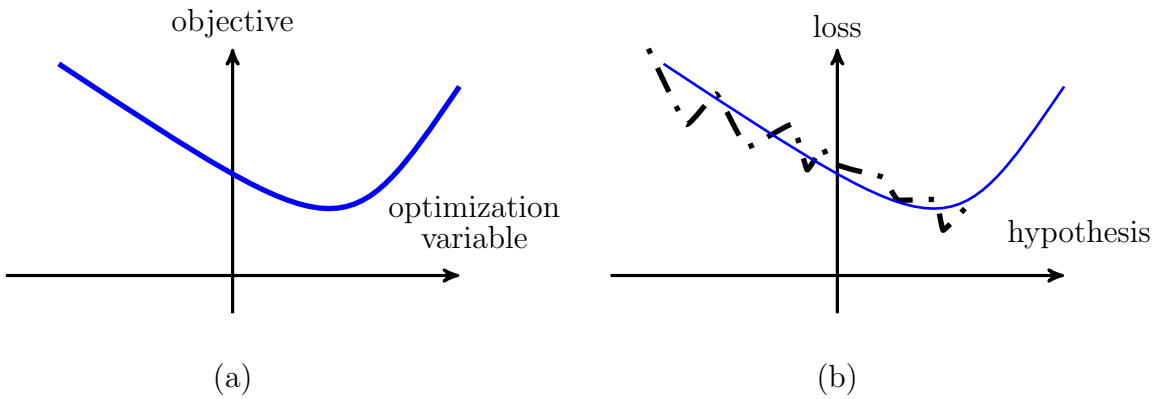


Figure 1.6: (a) A simple optimization problem consists of finding the values of an optimization variable that results in the minimum objective value. (b) ML methods learn (find) a hypothesis by minimizing a (average) loss. This average loss is a noisy version of the ultimate objective. This ultimate objective function is often defined as an expectation whose underlying probability distribution is unknown (see Section 2.3.4).

tools from the theory of optimization. What sets the optimization problems in ML apart from "plain vanilla" optimization problems (see Figure 1.6-(a)) is that we rarely have perfect access to the objective function to be minimized. ML methods learn a hypothesis by minimizing a noisy (or even incomplete) version (see Figure 1.6-(b)) of the ultimate objective function. The ultimate objective function for ML methods is often defined using an expectation over an unknown probability distribution for data points. Chapter 4 discusses methods that are based on estimating the objective function by empirical averages that are computed over a set of data points (which serve as a training set).

1.1.3 Theoretical Computer Science

Practical ML methods form a specific subclass of computing systems. Indeed, ML methods apply a sequence of computational operations to input data. The result of these computational operations are the predictions delivered to the user of the ML method. The interpretation of ML as computational systems allows to use tools from theoretical computer science to study the feasibility and intrinsic difficulty of ML problems. Even if a ML problem can be solved in theoretical sense, every practical ML method must fit the available computational infrastructure [113, 143].

The available computational resources, such as processor time, memory and communication bandwidth, can vary significantly between different infrastructures. One example for such a computational infrastructure is a single desktop computer. Another example

for a computational infrastructure is a cloud computing service which distributes data and computation over large networks of physical computers [95].

The focus of this book is on ML methods that can be understood as numerical optimization algorithms (see Chapter 4 and 5). Most of these ML methods amount to (a large number of) matrix operations such as matrix multiplication or matrix inversion [46]. Numerical linear algebra provides a vast algorithmic toolbox for the design of such ML methods [136, 135]. The recent success of ML methods in several application domains might be attributed to their efficient use of matrices to represent data and models. Using this representation allows us to implement the resulting ML methods using highly efficient hard- and software implementations for numerical linear algebra [48].

1.1.4 Information Theory

Information theory studies the problem of communication via noisy channels [140, 127, 27, 42]. Figure 1.7 depicts the most simple communication problem that consists of an information source that wishes communicate a message m over an imperfect (or noisy) channel to a receiver. The receiver tries to reconstruct (or learn) the original message based solely on the noisy channel output. Two main goals of information theory are (i) the characterization of conditions that allow reliable, i.e., nearly error-free, communication and (ii) the design of efficient transmitter (coding and modulation) and receiver (demodulation and decoding) methods.

It turns out that many concepts from information theory are very useful for the analysis and design of ML methods. As a point in case, Chapter 10 discusses the application of information-theoretic concepts to the design of explainable ML methods. On a more fundamental level, we can identify two core communication problems that arise within ML. These communication problems correspond, respectively, to the inference (making a prediction) and the learning (adjusting or improving the current hypothesis) step of a ML method (see Figure 1).

We can an interpret the inference step of ML as the problem of decoding the true label of a data point for which we only know its features. This communication problem is depicted in Figure 1.7-(b). Here the message to be communicated is the true label of a random data point. This data point is "communicated" over a channel that only passes through its features. The inference step within a ML method then tries to decode the original message (true label) from the channel output (features) resulting in the predicted label. A recent line of work used this communication problem to study deep learning methods [140].

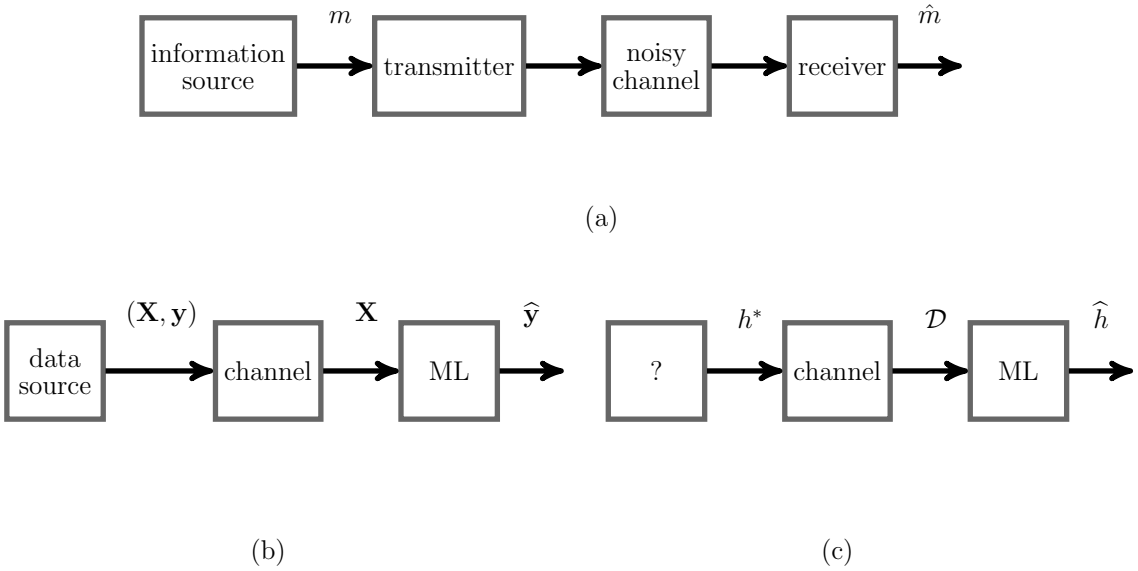


Figure 1.7: (a) A basic communication system involves an information source that emits a message m. The message is processed by some transmitter and passed through a noisy channel. The receiver tries to recover the original message m by computing the decoded message $\hat{m}$. (b) The inference step of ML (see Figure 1) corresponds to a communication problem with an information source emitting a data point with features $\mathbf{x}$ and label y. The ML method receives the features $\mathbf{x}$ and, in an effort to recover the true label y, computes the predicted label $\hat{y}$. (c) The learning or adaptation step of ML (see Figure 1) solves a communication problem with some source that selects a true (but unknown) hypothesis h^* as the message. The message is passed through an abstract channel that outputs a set $\mathcal{D}$ of labeled data points which are used as the training set by an ML method. The ML method tries to decode the true hypothesis resulting in the learnt the hypothesis $\widehat{h}$.

A second core communication problem of ML corresponds to the problem of learning (or adjusting) a hypothesis (see Figure 1.7-(c)). In this problem, the source selects some "true" hypothesis as message. This message is then communicated to an abstract channel that models the data generation process. The output of this abstract channel are data points in a training set $\mathcal{D}$ (see Chapter 4). The learning step of a ML method, such as empirical risk minimization (ERM) of Chapter 4, then amounts to the decoding of the message (true hypothesis) based on the channel output (training set). There is significant line or research that uses the communication problem in Figure 1.7-(c) to characterize the fundamental limits of ML problems and methods such as the minimum required number of training data points that makes learning feasible [155, 152, 122, 141, 71].

The relevance of information theoretic concepts and methods for ML is boosted by the recent trend towards distributed or federated ML [93, 130, 124, 123]. We can interpret federated learning (FL) applications as a specific type of network communication problems [42]. In particular, we can apply network coding techniques to the design and analysis of FL methods [42].

1.1.5 Probability Theory and Statistics

Consider the data points $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}$ depicted in Figure 1.2. Each data point represents some previous day that is characterized by its minimum and maximum daytime temperature as measured at a specific FMI weather observation station. It might be useful to interpret these data points as realizations of independent and identically distributed (i.i.d.) RVs with common (but typically unknown) probability distribution $p(\mathbf{z})$. Figure 1.8 extends the scatterplot in Figure 1.2 by adding a contour line of the underlying probability distribution $p(\mathbf{z})$ [9].

Probability theory offers principled methods for estimating a probability distribution from a set of data points (see Section 3.12). Let us assume we know (an estimate of) the (joint) probability distribution $p(\mathbf{z})$ of features and label of a data point $\mathbf{z} = \big(\mathbf{x}, y\big)$. A principled approach to predict the label value of a data point with features $\mathbf{x}$ is based on evaluating the conditional probability distribution $p(y = \hat{y}|\mathbf{x})$. The conditional probability distribution $p(\hat{y} = y|\mathbf{x})$ quantifies how likely it is that $\hat{y}$ is the actual label value of a data point. We can evaluate the quantity $p(\hat{y} = y|\mathbf{x})$ for any candidate value $\hat{y}$ as soon as we know the features $\mathbf{x}$ of this data point.

Depending on the performance criterion or loss function, the optimal prediction $\hat{y}$ is either given by the mode of $p(\hat{y} = y|\mathbf{x})$ its mean or some other characteristic value. It is important to

note that this probabilistic approach not only provides a specific prediction (point-estimate) but an entire distribution $p(\hat{y} = y|\mathbf{x})$ over possible predictions. This distribution allows to construct confidence measures, such as the variance, that can be provided along with the prediction.

Having a probabilistic model, in the form of a probability distribution $p(\mathbf{z})$, for the data arising in an ML application not only allows us to compute predictions for labels of data points. We can also use $p(\mathbf{z})$ to augment the available dataset by randomly drawing (generating) new data points from $p(\mathbf{z})$ (see Section 7.3). ML methods that rely on a probabilistic model for data are sometimes referred to as generative methods. A recently popularized class of generative methods, that uses models obtained from ANN, is known as generative adversarial networks [49].

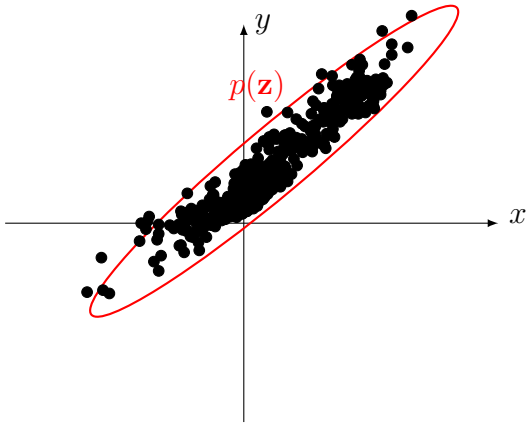


Figure 1.8: Each dot represents a data point $\mathbf{z} = \big(x, y\big)$ that is characterized by a numeric feature x and a numeric label y. We also indicate a contour-line of a probability distribution $p(\mathbf{z})$ that could be used to interpret data points as realizations of i.i.d. RVs with common probability distribution $p(\mathbf{z})$.

1.1.6 Artificial Intelligence

ML theory and methods are instrumental for the analysis and design of artificial intelligence (AI) [121]. An AI system, typically referred to as an agent, interacts with its environment by executing (choosing between different) actions. These actions influence the environment as well as the state of the AI agent. The behaviour of an AI system is determined by how the perceptions made about the environment are used to form the next action.

From an engineering point of view, AI aims at optimizing behaviour to maximize a long-term return. The optimization of behaviour is based solely on the perceptions made by the agent. Let us consider some application domains where AI systems can be used:

- a forest fire management system: perceptions given by satellite images and local observations using sensors or "crowd sensing" via some mobile application which allows humans to notify about relevant events; actions amount to issuing warnings and bans of open fire; return is the reduction of number of forest fires.

- a control unit for combustion engines: perceptions given by various measurements such as temperature, fuel consistency; actions amount to varying fuel feed and timing and the amount of recycled exhaust gas; return is measured in reduction of emissions.

- a severe weather warning service: perceptions given by weather radar; actions are preventive measures taken by farmers or power grid operators; return is measured by savings in damage costs (see `https://www.munichre.com/`)

- an automated benefit application system for the Finnish social insurance institute ("Kela"): perceptions given by information about application and applicant; actions are either to accept or to reject the application along with a justification for the decision; return is measured in reduction of processing time (applicants tend to prefer getting decisions quickly)

- a personal diet assistant: perceived environment is the food preferences of the app user and their health condition; actions amount to personalized suggestions for healthy and tasty food; return is the increase in well-being or the reduction in public spending for health-care.

- the cleaning robot Rumba (see Figure 1.9) perceives its environment using different sensors (distance sensors, on-board camera); actions amount to choosing different moving directions ("north", "south", "east", "west"); return might be the amount of cleaned floor area within a particular time period.

- personal health assistant: perceptions given by current health condition (blood values, weight,...), lifestyle (preferred food, exercise plan); actions amount to personalized suggestions for changing lifestyle habits (less meat, more walking,...); return is measured via the level of well-being (or the reduction in public spending for health-care).

- a government-system for a country: perceived environment is constituted by current economic and demographic indicators such as unemployment rate, budget deficit, age distribution,...; actions involve the design of tax and employment laws, public investment in infrastructure, organization of health-care system; return might be determined by the gross domestic product, the budget deficit or the gross national happiness (cf. `https://en.wikipedia.org/wiki/Gross_National_Happiness`).

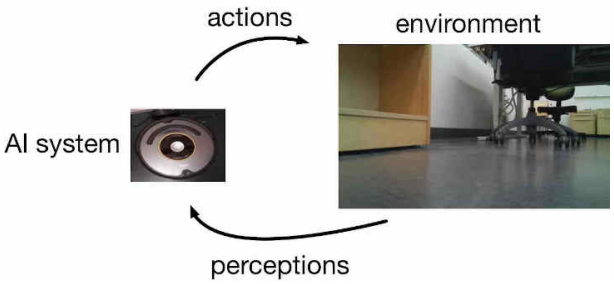


Figure 1.9: A cleaning robot chooses actions (moving directions) to maximize a long-term reward measured by the amount of cleaned floor area per day.

ML methods are used on different levels by an AI agent. On a lower level, ML methods help to extract the relevant information from raw data. ML methods are used to classify images into different categories which are then used an input for higher level functions of the AI agent.

ML methods are also used for higher level tasks of an AI agent. To behave optimally, an agent is required to learn a good hypothesis for how its behaviour affects its environment. We can think of optimal behaviour as a consequent choice of actions that might be predicted by ML methods.

What sets AI applications apart from more traditional ML application is that there is an strong interaction between ML method and the data generation process. Indeed, AI agents use the predictions of an ML method to select its next action which, in turn, influences the environment which generates new data points. The ML subfield of active learning studies methods that can influence the data generation [25].

Another characteristic of AI applications is that they typically allow ML methods to evaluate the quality of a hypothesis only in hindsight. Within a basic (supervised) ML application it is possible for a ML method to try out many different hypotheses on the same data point. These different hypotheses are then scored by their discrepancies with a known

correct predictions. In contrast to such passive ML applications, AI applications involve data points for which it is infeasible to determine the correct predictions.

Let us illustrate the above differences between ML and AI applications with the help of a self-driving toy car. The toy-car is equipped with a small onboard computer, camera, sensors and actors that allow to define the steering direction. Our goal is to program the onboard computer such that it implements an AI agent that optimally steers the toy-car. This AI application involves data points that represent the different (temporal) states of the toy car during its ride. We use a ML method to predict the optimal steering direction for the current state. The prediction for the optimal steering angle is obtained by a hypothesis map that reads a snapshot form an on-board camera. Since these predictions are used to actually steer the car, they influence the future data points (states) that will be obtained.

Note that we typically do not know the actual optimal steering direction for each possible state of the car. It is infeasible to let the toy car roam around along any possible path and then manually label each on-board camera snapshot with the optimal steering direction (see Figure 1.12). The usefulness of a prediction can be measured only in an indirect fashion by using some form of reward signal. Such a reward signal could be obtained from a distance sensor that allows to determine if the toy car reduced the distance to a target location.

1.2 Flavours of Machine Learning

ML methods read in data points which are generated within some application domain. An individual data point is characterized by various properties. We find it convenient to divide the properties of data points into two groups: features and labels (see Chapter 2.1). Features are properties that we measure or compute easily in an automated fashion. Labels are properties that cannot be measured easily and often represent some higher level fact (or quantity of interest) whose discovery often requires human experts.

Roughly speaking, ML aims at learning to predict (approximating or guessing) the label of a data point based solely on the features of this data point. Formally, the prediction is obtained as the function value of a hypothesis map whose input argument are the features of a data point. Since any ML method must be implemented with finite computational resources, it can only consider a subset of all possible hypothesis maps. This subset is referred to as the hypothesis space or model underlying a ML method. Based on how ML methods assess the quality of different hypothesis maps we distinguish three main flavours of ML: supervised, unsupervised and reinforcement learning.

1.2.1 Supervised Learning

The main focus of this book is on supervised ML methods. These methods use a training set that consists of labeled data points (for which we know the correct label values). We refer to a data point as labeled if its label value is known. Labeled data points might be obtained from human experts that annotate ("label") data points with their label values. There are marketplaces for renting human labelling workforce [132]. Supervised ML searches for a hypothesis that can imitate the human annotator and allows to predict the label solely from the features of a data point.

Figure 1.10 illustrates the basic principle of supervised ML methods. These methods learn a hypothesis with minimum discrepancy between its predictions and the true labels of the data points in the training set. Loosely speaking, supervised ML fits a curve (the graph of the predictor map) to labeled data points in a training set. For the actual implementing of this curve fitting we need a loss function that quantifies the fitting error. Supervised ML method differ in their choice for a loss function to measure the discrepancy between predicted label and true label of a data point.

While the principle behind supervised ML sounds trivial, the challenge of modern ML applications is the sheer amount of data points and their complexity. ML methods must process billions of data points with each single data points characterized by a potentially vast number of features. Consider data points representing social network users, whose features include all media that has been posted (videos, images, text). Besides the size and complexity of datasets, another challenge for modern ML methods is that they must be able to fit highly non-linear predictor maps. Deep learning methods address this challenge by using a computationally convenient representation of non-linear maps via artificial neural networks [48].

1.2.2 Unsupervised Learning

Some ML methods do not require knowing the label value of any data point and are therefore referred to as unsupervised ML methods. Unsupervised methods must rely solely on the intrinsic structure of data points to learn a good hypothesis. Thus, unsupervised methods do not need a teacher or domain expert who provides labels for data points (used to form a training set). Chapters 8 and 9 discuss two large families of unsupervised methods, referred to as clustering and feature learning methods.

Clustering methods group data points into few subsets or cluster. The data points within

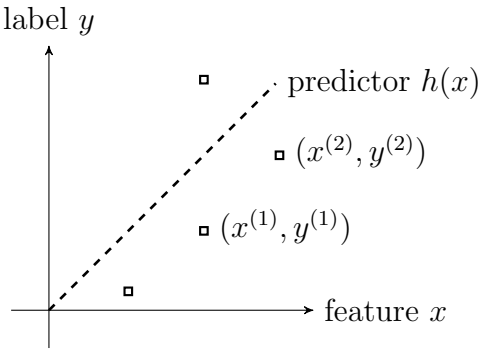


Figure 1.10: Supervised ML methods fit a curve to a set of data points (which serve as a training set). The curve represents a hypothesis out of some hypothesis space or model. The fitting error (or training error) is measured using a loss function. Different ML methods use different combinations of model and loss function.

the same cluster should be more similar with each other than with data points outside the cluster (see Figure 1.11). Feature learning methods determine numeric features such that data points can be processed efficiently using these features. Two important applications of feature learning are dimensionality reduction and data visualization.

1.2.3 Reinforcement Learning

In general, ML methods use a loss function to evaluate and compare different hypotheses. The loss function assigns a (typically non-negative) loss value to a pair of a data point and a hypothesis. ML methods search for a hypothesis, out of (typically large) hypothesis space, that incurs minimum loss for any data point. Reinforcement learning (RL) studies applications where the predictions obtained by a hypothesis influences the generation of future data points. RL applications involve data points that represent the states of a programmable system (an AI agent) at different time instants. The label of such a data point has the meaning of an optimal action that the agent should take in a given state. Similar to unsupervised ML, RL methods often must learn a hypothesis without having access to any labeled data point.

In stark contrast to supervised and unsupervised ML methods, RL methods cannot evaluate the loss function for different choices of a hypothesis. Consider a RL method that has to predict the optimal steering angle of a car. Naturally, we can only evaluate the usefulness specific combination of predicted label (steering angle) and the current state of the car. It is impossible to try out two different hypotheses at the same time as the car cannot follow

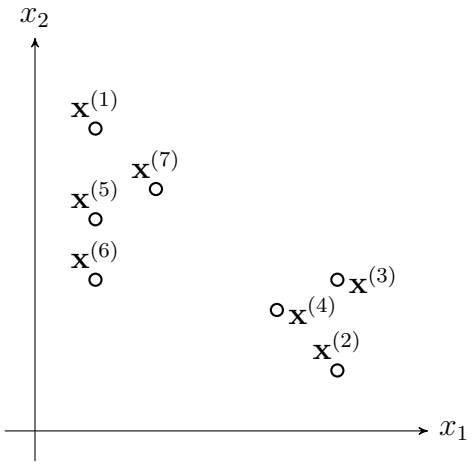


Figure 1.11: Clustering methods learn to predict the cluster (or group) assignments of data points based solely on their features. Chapter 8 discusses clustering methods that are unsupervised in the sense of not requiring the knowledge of the true cluster assignment of any data point.

two different steering angles (obtained by the two hypotheses) at the same time.

Mathematically speaking, RL methods can evaluate the loss function only point-wise for the current hypothesis that has been used to obtain the most recent prediction. These point-wise evaluations of the loss function are typically implemented by using some reward signal [137]. Such a reward signal might be obtained from a sensing device and allows to quantify the usefulness of the current hypothesis.

One important application domain for RL methods is autonomous driving (see Figure 1.12). Consider data points that represent individual time instants $t = 0, 1, \ldots$ during a car ride. The features of the tth data point are the pixel intensities of an on-board camera snapshot taken at time t. The label of this data point is the optimal steering direction at time t to maximize the distance between the car and any obstacle. We could use a ML method to learn hypothesis for predicting the optimal steering direction solely from the pixel intensities in the on-board camera snapshot. The loss incurred by a particular hypothesis is determined from the measurement of a distance sensor after the car moved along the predicted direction. We can evaluate the loss only for the hypothesis that has actually been used to predict the optimal steering direction. It is impossible to evaluate the loss for other predictions of the optimal steering direction since the car already moved on.

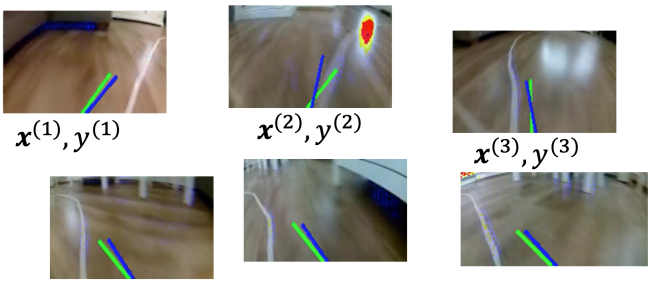


Figure 1.12: Autonomous driving requires to predict the optimal steering direction (label) based on an on-board camera snapshot (features) in each time instant. RL methods sequentially adjust a hypothesis for predicting the steering direction from the snapshot. The quality of the current hypothesis is evaluated by the measurement of a distance sensor (to avoid collisions with obstacles).

1.3 Organization of this Book

Chapter 2 introduces the notions of data, a model and a loss function as the three main components of ML. We will also highlight some of the computational and statistical aspects that might guide the design choices arising for these three components. A guiding theme of this book is the depiction of ML methods as combinations of specific design choices for data representation, the model and the loss function. Put differently, we aim at mapping out the vast landscape of ML methods in an abstract three-dimensional space spanned by the three dimensions: data, model and loss.

Chapter 3 details how several well-known ML methods are obtained by specific design choices for data (representation), model and loss function. Examples range from basic linear regression (see Section 3.1) via support vector machine (SVM) (see Section 3.7) to deep reinforcement learning (see Section 3.14).

Chapter 4 discusses a principle approach to combine the three components within a practical ML method. In particular, Chapter 4 explains how a simple probabilistic model for data lends naturally to the principle of ERM. This principle translates the problem of learning into an optimization problem. ML methods based on the ERM are therefore a special class of optimization methods. The ERM principle can be interpreted as a precise mathematical formulation of the "learning by trial and error" paradigm.

Chapter 5 discusses a family of iterative methods for solving the ERM problem introduced in Chapter 4. These methods use the concept of a gradient to locally approximate the objective function used in ERM. Gradient-based methods are widely used within deep

learning methods to learn useful weights for large ANN (see Section 3.11 and [48]).

The ERM principle of Chapter 4 requires a hypothesis to accurately predict the labels of data points in a training set. However, the ultimate goal of ML is to learn a hypothesis that delivers accurate predications for any data point and not only the training set. Chapter 6 discusses some basic validation techniques that allow to probe a hypothesis outside the training set that has been used to learn (optimize) this hypothesis via ERM. Validation techniques are instrumental for model selection, i.e., to choose the best model from a given set of candidate models. Chapter 7 presents regularization techniques that aim at replacing the training error of a candidate hypothesis with an estimate (or approximation) of its average loss incurred for data points outside the training set.

The focus of Chapters 3 - 7 is on supervised ML methods that require a training set of labeled data points. Chapters 8 and 9 are devoted to unsupervised ML methods which do not require any labeled data. Chapter 8 discusses clustering methods that partition data points into coherent groups which are referred to as clusters. Chapter 9 discusses methods that automatically determine the most relevant characteristics (or features) of a data point. This chapter also highlights the importance of using only the most relevant features of a data point, and to avoid irrelevant features, to reduce computational complexity and improve the accuracy of ML methods (such as those discussed in Chapter 3).

The success of ML methods becomes increasingly dependent on their explainability or transparency for the user of the ML method [54, 92]. The explainability of a ML method and its predictions typically depends on the background knowledge of the user which might vary significantly. Chapter 10 discusses two different approaches to obtain personalized explainable ML. These techniques use a feedback signal, which is provided for the data points in a training set, to either compute personalized explanations for a given ML method or to choose models that are intrinsically explainable to a specific user.

Prerequisites. We assume familiarity with basic notions and concepts of linear algebra, real analysis, and probability theory [136, 119]. For a brief review of those concepts, we recommend [48, Chapter 2-4] and the references therein. A main goal of this book is to develop the basic ideas and principles behind ML methods using a minimum of probability theory. However, some rudimentary knowledge about the concept of expectations, probability density function of a continuous (real-valued) RV and probability mass function of a discrete RV is helpful [9, 50].

Chapter 2

Three Components of ML

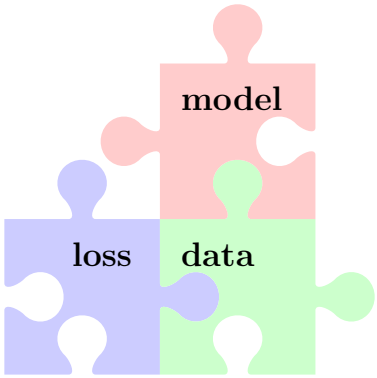


Figure 2.1: ML methods fit a model to data via minimizing a loss function (see Figure 1). We obtain a variety of ML methods from different design choices for the model, loss function and data representation (see Chapter 3). A principled approach to combine these three components is ERM (see Chapter 4).

This book portrays ML as combinations of three components as depicted in Figure 2.1. These components are

- data as collections of individual data points that are characterized by features (see Section 2.1.1) and labels (see Section 2.1.2)
- a model or hypothesis space that consists of computationally feasible hypothesis maps from feature space to label space (see Section 2.2)
- a loss function (see Section 2.3) to measure the quality of a hypothesis map.

A ML problem involves specific design choices for data points, its features and labels, the hypothesis space and the loss function to measure the quality of a particular hypothesis. Similar to ML problems (or applications), we can also characterize ML methods as combinations of these three components. This chapter discusses in some depth each of the above ML components and their combination within some widely-used ML methods [112].

We detail in Chapter 3 how some of the most popular ML methods, including linear regression (see Section 3.1) as well as deep learning methods (see Section 3.11), are obtained by specific design choices for the three components. Chapter 4 will then introduce ERM as a main principle for how to operationally combine the individual ML components. Within the ERM principle, ML problems become optimization problems and ML methods become optimization methods.

2.1 The Data

Data as Collections of Data points. Maybe the most important component of any ML problem (and method) is data. We consider data as collections of individual data points which are atomic units of "information containers". Data points can represent text documents, signal samples of time series generated by sensors, entire time series generated by collections of sensors, frames within a single video, random variables, videos within a movie database, cows within a herd, trees within a forest, or forests within a collection of forests. Mountain hikers might be interested in data points that represent different hiking tours (see Figure 2.2).

Figure 2.2: A snapshot taken at the beginning of a mountain hike.

We use the concept of data points in a very abstract and therefore highly flexible manner.

Data points can represent very different types of objects that arise in fundamentally different application domains. For an image processing application it might be useful to define data points as images. When developing a recommendation system we might define data points to represent customers. In the development of new drugs we might use data points to represent different diseases. Ultimately, the choice or definition of data points is a design choice. We might refer to the task of finding a useful definition of data points as "data point engineering".

One practical requirement for a useful definition of data points is that we should have access to many of them. Many ML methods construct estimates for a quantity of interest (such as a prediction or forecast) by averaging over a set of reference (or training) data points. These estimates become more accurate for an increasing number of data points used for computing the average. A key property of a dataset is the number m of individual data points it contains. The number of data points within a dataset is also referred to as the sample size. From a statistical point of view, the larger the sample size m the better. However, there might be restrictions on computational resources (such as memory size) that limit the maximum sample size m that can be processed.

For most applications, it is impossible to have full access to every single microscopic property of a data point. Consider a data point that represents a vaccine. A full characterization of such a data point would require to specify its chemical composition down to level of molecules and atoms. Moreover, there are properties of a vaccine that depend on the patient who received the vaccine.

We find it useful to distinguish between two different groups of properties of a data point. The first group of properties is referred to as features and the second group of properties is referred to as labels. Roughly speaking, features are low-level properties of a data point that can be measured or computed easily in an automated fashion. In contract, labels are high-level properties of a data points that represent some quantity of interest. Determining the label value of a data point often requires human labour, e.g., a domain expert who has to examine the data point. Some widely used synonyms for features are "covariate","explanatory variable", "independent variable", "input (variable)", "predictor (variable)" or "regressor" [52, 30, 38]. Some widely used synonyms for the label of a data point are "response variable", "output variable" or "target" [52, 30, 38].

We will discuss the concepts of features and labels in somewhat more detail in Sections 2.1.1 and 2.1.2. However, we would like to point out that the distinction between features and labels is blurry. The same property of a data point might be used as a feature in one

application, while it might be used as a label in another application. Let us illustrate this blurry distinction between features and labels using the problem of missing data.

Assume we have a list of data points each of which is characterized by several properties that could be measured easily in principles (by sensors). These properties would be first candidates for being used as features of the data points. However, some of these properties are unknown (missing) for a small set of data points (e.g., due to broken sensors). We could then define the properties which are missing for some data points as labels and try to predict these labels using the remaining properties (which are known for all data points) as features. The task of determining missing values of properties that could be measured easily in principle is referred to as imputation [1].

Missing data might also arise in image processing applications. Consider data points being images (snapshots) generated by a smartphone camera. Maybe the most intuitive choice for the features of a (bitmap) image are the colour intensities for each pixel (see Figure 2.5). Due to hardware failures some of the image pixels might be corrupted or (their colour intensities) even completely missing. We could then try to use to learn to predict the colour intensities of a pixel based on the colour intensities of the neighbouring pixels. To this end, we might define new data points as small rectangular regions (patches) of the image and use the colour intensity of the centre pixel ("target pixel") as the label of such a patch.

Figure 2.3 illustrates two key properties of a dataset. The first property is the sample size m, i.e., the number of individual data points that constitute the dataset. The second key property of is the number n of features that are used to characterize an individual data point. The behaviour of ML methods often depends crucially on the ratio m/n. The performance of ML methods typically improves with increasing m/n. As a rule of thumb, we should use datasets for which $m/n \gg 1$. We will make the informal condition $m/n \gg 1$ more precise in Chapter 6.

2.1.1 Features

Similar to the choice (or definition) of data points with an ML application, also the choice of which properties to be used as their features is a design choice. In general, features are (low-level) properties of a data point that can be computed or measured easily. This is obviously a highly informal characterization since there is no universal criterion for the difficulty of computing of measuring a specific property of data points. The task of choosing which properties to use as features of data points might be the most challenging part in the

n

m

Year	m	d	Time	precp	snow	airtmp	mintmp	maxtmp
2020	1	2	00:00	0,4	55	2,5	-2	4,5
2020	1	3	00:00	1,6	53	0,8	-0,8	4,6
2020	1	4	00:00	0,1	51	-5,8	-11,1	-0,7
2020	1	5	00:00	1,9	52	-13,5	-19,1	-4,6
2020	1	6	00:00	0,6	52	-2,4	-11,4	-1
2020	1	7	00:00	4,1	52	0,4	-2	1,3

Figure 2.3: Two key properties of a dataset are the number (sample size) m of individual data points that constitute the dataset and the number n of features used to characterize individual data points. The behaviour of ML methods typically depends crucially on the ratio m/n.

application of ML methods. Chapter 9 discusses feature learning methods that automate (to some extend) the construction of good features.

In some application domains there is a rather natural choice for the features of a data point. For data points representing audio recording (of a given duration) we might use the signal amplitudes at regular sampling instants (e.g., using sampling frequency 44 kHz) as features. For data points representing images it seems natural to use the colour (red, green and blue) intensity levels of each pixel as a feature (see Figure 2.5).

The feature construction for images depicted in Figure 2.5 can be extended to other types of data points as long as they can be visualized efficiently [40]. Audio recordings are typically available a sequence of signal amplitudes a_t collected regularly at time instants $t = 1, \ldots, n$ with sampling frequency ≈ 44 kHz. From a signal processing perspective, it seems natural to directly use the signal amplitudes as features, $x_j = a_j$ for $j = 1, \ldots, n$. However, another choice for the features would be the pixel RGB values of some visualization of the audio recording.

Figure 2.4 depicts two possible visualizations of an audio signal. The first visualization is obtained from a line plot of the signal amplitudes as a function of time t. Another visualization of an audio recording is obtained from an intensity plot of its spectrogram[13, 90]. We can then use the pixel RGB intensities of these visualizations as the features for an audio recording. Using this trick we can transform any ML method for image data to an ML method for audio data. We can use the scatterplot of a data set to use ML methods for

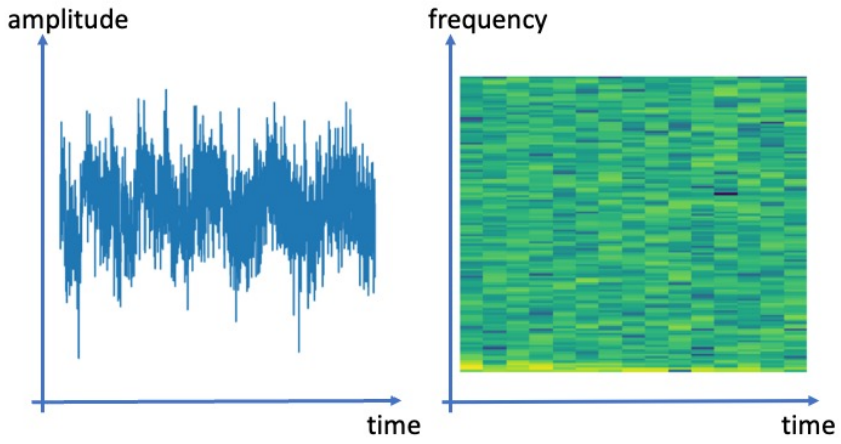


Figure 2.4: Two visualizations of a data point that represents an audio recording. The left figure shows a line plot of the audio signal amplitudes. The right figure shows a spectogram of the audio recording.

image segmentation to cluster the dataset(see Chapter 8).

Many important ML application domains generate data points that are characterized by several numeric features $x_1, \ldots, x_n$. We represent numeric features by real numbers $x_1, \ldots, x_n \in \mathbb{R}$ which might seem impractical. Indeed, digital computers cannot store a real number exactly as this would require an infinite number of bits. However, numeric linear algebra soft- and hardware allows to approximate real numbers with sufficient accuracy.

The majority of ML methods discussed in this book assume that data points are characterized by real-valued features. Section 9.3 discusses methods for constructing numeric features of data points whose natural representation is non-numeric.

We assume that data points arising in a given ML application are characterized by the same number n of individual features $x_1 \ldots, x_n$. It is convenient to stack the individual features of a data point into a single feature vector

$$\mathbf{x} = \big(x_1, \ldots, x_n\big)^T.$$

Each data point is then characterized by its feature vector $\mathbf{x}$. Note that stacking the features of a data point into a column vector $\mathbf{x}$ is pure convention. We could also arrange the features as a row vector or even as a matrix, which might be even more natural for features obtained by the pixels of an image (see Figure 2.5).

We refer to the set of possible feature vectors of data points arising in some ML appli-

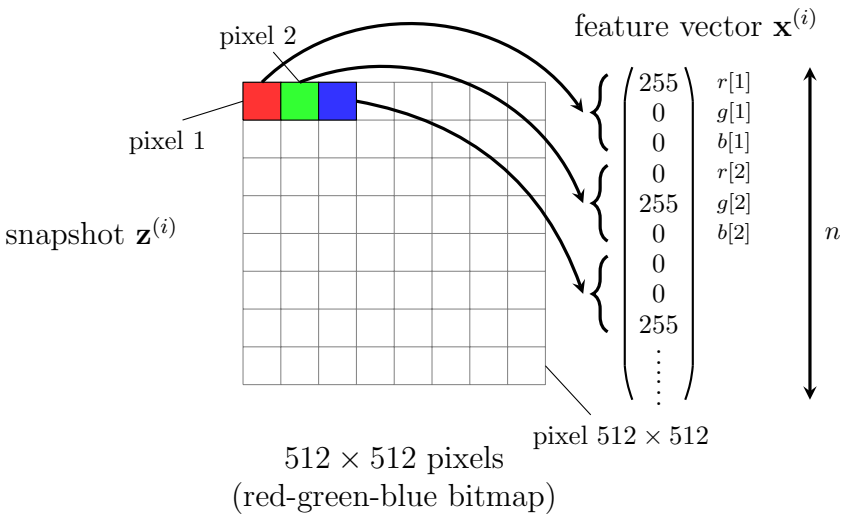


Figure 2.5: If the snapshot $\mathbf{z}^{(i)}$ is stored as a 512×512 RGB bitmap, we could use as features $\mathbf{x}^{(i)} \in \mathbb{R}^n$ the red-, green- and blue component of each pixel in the snapshot. The length of the feature vector would then be $n = 3 \times 512 \times 512 \approx 786000$.

cation as the feature space and denote it as $\mathcal{X}$. The feature space is a design choice as it depends on what properties of a data point we use as its features. This design choice should take into account the statistical properties of the data as well as the available computational infrastructure. If the computational infrastructure allows for efficient numerical linear algebra, then using $\mathcal{X} = \mathbb{R}^n$ might be a good choice.

The Euclidean space $\mathbb{R}^n$ is an example of a feature space with a rich geometric and algebraic structure [119]. The algebraic structure of $\mathbb{R}^n$ is defined by vector addition and multiplication of vectors with scalars. The geometric structure of $\mathbb{R}^n$ is obtained by the Euclidean norm as a measure for the distance between two elements of $\mathbb{R}^n$. The algebraic and geometric structure of $\mathbb{R}^n$ often enables an efficient search over $\mathbb{R}^n$ to find elements with desired properties. Chapter 4.3 discusses examples of such search problems in the context of learning an optimal hypothesis.

Modern information-technology, including smartphones or wearables, allows us to measure a huge number of properties about data points in many application domains. Consider a data point representing the book author "Alex Jung". Alex uses a smartphone to take roughly five snapshots per day (sometimes more, e.g., during a mountain hike) resulting in more than 1000 snapshots per year. Each snapshot contains around 10^6 pixels whose greyscale levels we can use as features of the data point. This results in more than 10^9

features (per year!).

As indicated above, many important ML applications involve data points represented by very long feature vectors. To process such high-dimensional data, modern ML methods rely on concepts from high-dimensional statistics [17, 151]. One such concept from high-dimensional statistics is the notion of sparsity. Section 3.4 discusses methods that exploit the tendency of high-dimensional data points, which are characterized by a large number n of features, to concentrate near low-dimensional subspaces in the feature space [147].

At first sight it might seem that "the more features the better" since using more features might convey more relevant information to achieve the overall goal. However, as we discuss in Chapter 7, it can be detrimental for the performance of ML methods to use an excessive amount of (irrelevant) features. Computationally, using too many features might result in prohibitive computational resource requirements (such as processing time). Statistically, each additional feature typically introduces an additional amount of noise (due to measurement or modelling errors) which is harmful for the accuracy of the ML method.

It is difficult to give a precise and broadly applicable characterization of the maximum number of features that should be used to characterize the data points. As a rule of thumb, the number m of (labeled) data points used to train a ML method should be much larger than the number n of numeric features (see Figure 2.3). The informal condition $m/n \gg 1$ can be ensured by either collecting a sufficiently large number m of data points or by using a sufficiently small number n of features. We next discuss implementations for each of these two complementary approaches.

The acquisition of (labeled) data points might be costly, requiring human expert labour. Instead of collecting more raw data, it might be more efficient to generate new artificial (synthetic) data via data augmentation techniques. Section 7.3 shows how intrinsic symmetries in the data can be used to augment the raw data with synthetic data. As an example for an intrinsic symmetry of data, consider data points being images. We assign each image the label $y = 1$ if it shows a cat and $y = -1$ otherwise. For each image with known label we can generate several augmented (additional) images with the same label. These additional images might be obtained by simple image transformation such as rotations or re-scaling (zoom-in or zoom-out) that do not change the depicted objects (the meaning of the image). Chapter 7 shows that some basic regularization techniques can be interpreted as an implicit form of data augmentation.

The informal condition $m/n \gg 1$ can also be ensured by reducing the number n of features used to characterize data points. In some applications, we might use some domain

knowledge to choose the most relevant features. For other applications, it might be difficult to tell which quantities are the best choice for features. Chapter 9 discusses methods that learn, based on some given dataset, to determine a small number of relevant features of data points.

Beside the available computational infrastructure, also the statistical properties of datasets must be taken into account for the choices of the feature space. The linear algebraic structure of $\mathbb{R}^n$ allows us to efficiently represent and approximate datasets that are well aligned along linear subspaces. Section 9.2 discusses a basic method to optimally approximate datasets by linear subspaces of a given dimension. The geometric structure of $\mathbb{R}^n$ is also used in Chapter 8 to decompose a dataset into few groups or clusters that consist of similar data points.

Throughout this book we will mainly use the feature space $\mathbb{R}^n$ with dimension n being the number of features of a data point. This feature space has proven useful in many ML applications due to availability of efficient soft- and hardware for numerical linear algebra. Moreover, the algebraic and geometric structure of $\mathbb{R}^n$ reflect the intrinsic structure of data arising in many important application domains. This should not be too surprising as the Euclidean space has evolved as a useful mathematical abstraction of physical phenomena [77].

In general there is no mathematically correct choice for which properties of a data point to be used as its features. Most application domains allow for some design freedom in the choice of features. Let us illustrate this design freedom with a personalized health-care applications. This application involves data points that represent audio recordings with the fixed duration of three seconds. These recordings are obtained via smartphone microphones and used to detect coughing [6].

2.1.2 Labels

Besides its features, a data point might have a different kind of properties. These properties represent a higher-level fact or quantity of interest that is associated with the data point. We refer to such properties of a data point as its label (or "output" or "target") and typically denote it by y (if it is a single number) or by $\mathbf{y}$ (if it is a vector of different label values, such as in multi-label classification). We refer to the set of all possible label values of data points arising in a ML application is the label space $\mathcal{Y}$. In general, determining the label of a data point is more difficult (to automate) compared to determining its features. Many ML methods revolve around finding efficient ways to predict (estimate or approximate) the label of a data point based solely on its features.

The distinction of data point properties into labels and features is blurry. Roughly speaking, labels are properties of data points that might only be determined with the help of human experts. For data points representing humans we could define its label y as an indicator if the person has flu ($y = 1$) or not ($y = 0$). This label value can typically only be determined by a physician. However, in another application we might have enough resources to determine the flu status of any person of interest and could use it as a feature that characterizes a person.

Consider a data point that represents a hike, at the start of which the snapshot in Figure 2.2 has been taken. The features of this data point could be the red, green and blue (RGB) intensities of each pixel in the snapshot in Figure 2.2. We stack these RGB values into a vector $\mathbf{x} \in \mathbb{R}^n$ whose length n is three times the number of pixels in the image. The label y associated with a data point (a hike) could be the expected hiking time to reach the mountain in the snapshot. Alternatively, we could define the label y as the water temperature of the lake that is depicted in the snapshot.

Numeric Labels - Regression. For a given ML application, the label space $\mathcal{Y}$ contains all possible label values of data points. In general, the label space is not just a set of different elements but also equipped (algebraic or geometric) structure. To obtain efficient ML methods, we should exploit such structure. Maybe the most prominent example for such a structured label space are the real numbers $\mathcal{Y} = \mathbb{R}$. This label space is useful for ML applications involving data points with numeric labels that can be modelled by real numbers. ML methods that aim at predicting a numeric label are referred to as regression methods.

Categorical Labels - Classification. Many important ML applications involve data points whose label indicate the category or class to which data points belongs to. ML methods that aim at predicting such categorical labels are referred to as classification methods. Examples for classification problems include the diagnosis of tumours as benign or malefic cent, the classification of persons into age groups or detecting the current floor conditions ("grass", "tiles" or "soil") for a mower robot.

The most simple type of a classification problems is a binary classification problem. Within binary classification, each data point belongs to exactly one out of two different classes. Thus, the label of a data point takes on values from a set that contains two different elements such as $\{0, 1\}$ or $\{-1, 1\}$ or {"shows cat", "shows no cat"}.

We speak of a multi-class classification problem if data points belong to exactly one out of more than two categories (e.g., image categories "no cat shown" vs. "one cat shown" and "more than one cat shown"). If there are K different categories we might use the label

values $\{1, 2, \ldots, K\}$.

There are also applications where data points can belong to several categories simultaneously. For example, an image can be cat image and a dog image at the same time if it contains a dog and a cat. Multi-label classification problems and methods use several labels $y_1, y_2, \ldots$, for different categories to which a data point can belong to. The label y_j represents the jth category and its value is $y_j = 1$ if the data point belongs to the j-th category and $y_j = 0$ if not.

Ordinal Labels. Ordinal label values are somewhat in between numeric and categorical labels. Similar to categorical labels, ordinal labels take on values from a finite set. Moreover, similar to numeric labels, ordinal labels take on values from an ordered set. For an example for such an ordered label space, consider data points representing rectangular areas of size 1 km by 1 km. The features $\mathbf{x}$ of such a data point can be obtained by stacking the RGB pixel values of a satellite image depicting that area (see Figure 2.5). Beside the feature vector, each rectangular area is characterized by a label $y \in \{1, 2, 3\}$ where

- $y = 1$ means that the area contains no trees.
- $y = 2$ means that the area is partially covered by trees.
- $y = 3$ means that the are is entirely covered by trees.

Thus we might say that label value $y = 2$ is "larger" than label value $y = 1$ and label value $y = 3$ is "larger" than label value $y = 2$.

The distinction between regression and classification problems and methods is somewhat blurry. Consider a binary classification problem based on data points whose label y takes on values -1 or 1. We could turn this into a regression problem by using a new label y' which is defined as the confidence in the label y being equal to 1. On the other hand, given a prediction $\hat{y'}$ for the numeric label $y' \in \mathbb{R}$ we can obtain a prediction $\hat{y}$ for the binary label $y \in \{-1, 1\}$ by setting $\hat{y} := 1$ if $\hat{y'} \geq 0$ and $\hat{y} := -1$ otherwise. A prominent example for this link between regression and classification is logistic regression which is discussed in Section 3.6. Logistic regression is a binary classification method that uses the same model as linear regression but a different loss function.

We refer to a data point as being *labeled* if, besides its features $\mathbf{x}$, the value of its label y is known. The acquisition of labeled data points typically involves human labour, such as verifying if an image shows a cat. In other applications, acquiring labels might require sending out a team of marine biologists to the Baltic sea [131], to run a particle physics

experiment at the European organization for nuclear research (CERN) [18], or to conduct animal trials in pharmacology [43].

Let us also point out online market places for human labelling workforce [99]. These market places, allow to upload data points, such as collections of images or audio recordings, and then offer an hourly rate to humans that label the data points. This labeling work might amount to marking images that show a cat.

Many applications involve data points whose features can be determined easily, but whose labels are known for few data points only. Labeled data is a scarce resource. Some of the most successful ML methods have been devised in application domains where label information can be acquired easily [55]. ML methods for speech recognition and machine translation can make use of massive labeled datasets that are freely available [78].

In the extreme case, we do not know the label of any single data point. Even in the absence of any labeled data, ML methods can be useful for extracting relevant information from features only. We refer to ML methods which do not require any labeled data points as "unsupervised" ML methods. We discuss some of the most important unsupervised ML methods in Chapter 8 and Chapter 9).

ML methods learn (or search for) a "good" predictor $h : \mathcal{X} \rightarrow \mathcal{Y}$ which takes the features $\mathbf{x} \in \mathcal{X}$ of a data point as its input and outputs a predicted label (or output, or target) $\hat{y} = h(\mathbf{x}) \in \mathcal{Y}$. A good predictor should be such that $\hat{y} \approx y$, i.e., the predicted label $\hat{y}$ is close (with small error $\hat{y} - y$) to the true underlying label y.

2.1.3 Scatterplot

Consider data points characterized by a single numeric feature x and single numeric label y. To gain more insight into the relation between the features and label of a data point, it can be instructive to generate a scatterplot as shown in Figure 1.2. A scatterplot depicts the data points $\mathbf{z}^{(i)} = (x^{(i)}, y^{(i)})$ in a two-dimensional plane with the axes representing the values of feature x and label y.

The visual inspection of a scatterplot might suggest potential relationships between feature x (minimum daytime temperature) and label y (maximum daytime temperature). From Figure 1.2, it seems that there might be a relation between feature x and label y since data points with larger x tend to have larger y. This makes sense since having a larger minimum daytime temperature typically implies also a larger maximum daytime temperature.

To construct a scatterplot for data points with more than two features we can use feature learning methods (see Chapter 9). These methods transform high-dimensional data points,

having billions of raw features, to three or two new features. These new features can then be used as the coordinates of the data points in a scatterplot.

2.1.4 Probabilistic Models for Data

A powerful idea in ML is to interpret each data points as the realization of a RV. For ease of exposition let us consider data points that are characterized by a single feature x. The following concepts can be extended easily to data points characterized by a feature vector $\mathbf{x}$ and a label y.

One of the most basic examples of a probabilistic model for data points in ML is the "i.i.d. assumption". This assumption interprets data points $x^{(1)}, \ldots, x^{(m)}$ as realizations of statistically independent RVs with the same probability distribution $p(x)$. It might not be immediately clear why it is a good idea to interpret data points as realizations of RVs with the common probability distribution $p(x)$. However, this interpretation allows us to use the properties of the probability distribution to characterize overall properties of entire datasets, i.e., large collections of data points.

The probability distribution $p(x)$ underlying the data points within the i.i.d. assumption is either known (based on some domain expertise) or estimated from data. It is often enough to estimate only some parameters of the distribution $p(x)$. Section 3.12 discusses a principled approach to estimate the parameters of a probability distribution from a given set of data points. This approach is sometimes referred to as maximum likelihood and aims at finding (parameter of) a probability distribution $p(x)$ such that the probability (density) of observing the given data points is maximized [85, 76, 9].

Two of the most basic and widely used parameters of a probability distribution $p(x)$ are the expected value or mean [11]

$$\mu_x = \mathbb{E}\{x\} := \int_{x'} x' p(x') dx'$$

and the variance

$$\sigma_x^2 := \mathbb{E}\big\{\big(x - \mathbb{E}\{x\}\big)^2\big\}.$$

These parameters can be estimated using the sample mean (average) and sample variance,

$$\hat{\mu}_x := (1/m) \sum_{i=1}^{m} x^{(i)} \text{ , and}$$
$$\widehat{\sigma_x^2} := (1/m) \sum_{i=1}^{m} \big(x^{(i)} - \hat{\mu}_x\big)^2. \tag{2.1}$$

The sample mean and sample variance (2.1) are the maximum likelihood estimators for the mean and variance of a normal (Gaussian) distribution $p(x)$ (see [12, Chapter 2.3.4]).

Most of the ML methods discussed in this book are motivated by an i.i.d. assumption. It is important to note that this i.i.d. assumption is only a modelling assumption (or hypothesis). There is no means to verify if an arbitrary set of data points are "exactly" realizations of i.i.d. RVs. There are principled statistical methods (hypothesis tests) that allow to verify if a given set of data point can be well approximated as realizations of i.i.d. RVs [88]. Alternatively, we can enforce the i.i.d. assumption if we generate synthetic data using a random number generator. Such synthetic i.i.d. data points could be obtained by sampling algorithms that incrementally build a synthetic dataset by adding randomly chosen raw data points [32].

2.2 The Model

Consider some ML application that generates data points, each characterized by features $\mathbf{x} \in \mathcal{X}$ and label $y \in \mathcal{Y}$. The goal of a ML method is to learn a hypothesis map $h : \mathcal{X} \to \mathcal{Y}$ such that

$$y \approx \underbrace{h(\mathbf{x})}_{\hat{y}} \text{ for any data point.} \tag{2.2}$$

The informal goal (2.2) will be made precise in several aspects throughout the rest of our book. First, we need to quantify the approximation error (2.2) incurred by a given hypothesis map h. Second, we need to make precise what we actually mean by requiring (2.2) to hold for "any" data point. We solve the first issue by the concept of a loss function in Section 2.3. The second issue is then solved in Chapter 4 by using a simple probabilistic model for data.

Let us assume for the time being that we have found a reasonable hypothesis h in the sense of (2.2). We can then use this hypothesis to predict the label of any data point for which we know its features. The prediction $\hat{y} = h(\mathbf{x})$ is obtained by evaluating the hypothesis for the features $\mathbf{x}$ of a data point (see Figure 2.6 and 2.7). We refer to a hypothesis map

also as a predictor map since it is used or compute the prediction $\hat{y}$ of the label y.

For ML problems using a finite label space $\mathcal{Y}$ (e.g, $\mathcal{Y} = \{-1, 1\}$), we refer to a hypothesis map also as a classifier. Given a finite label space $\mathcal{Y}$, we can characterize a particular classifier $h(\mathbf{x})$ using its different decision regions

$$\mathcal{R}_a := \big\{\mathbf{x} \in \mathbb{R}^n : h(\mathbf{x}) = a\big\} \subseteq \mathcal{X}. \tag{2.3}$$

Each label value $a \in \mathcal{Y}$ is associated with a specific decision region $\mathcal{R}_a$. For a given label value $a \in \mathcal{Y}$, the decision region $\mathcal{R}_a$ is constituted by all feature vectors $\mathbf{x} \in \mathcal{X}$ which are mapped to this label value, $h(\mathbf{x}) = a$.

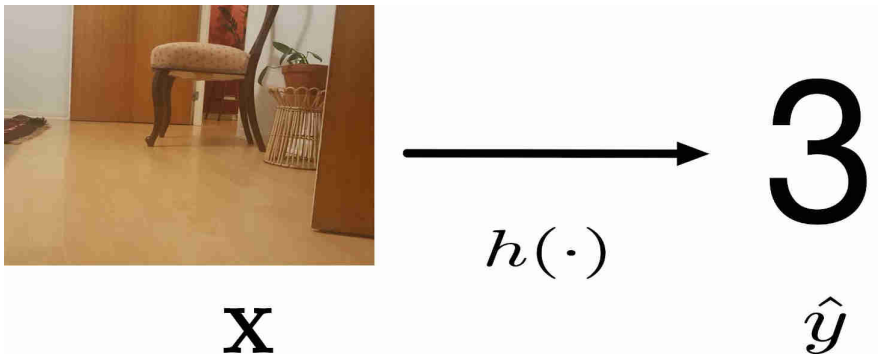


Figure 2.6: A hypothesis (or predictor) h maps features $\mathbf{x} \in \mathcal{X}$, of an on-board camera snapshot, to the prediction $\hat{y} = h(\mathbf{x}) \in \mathcal{Y}$ for the coordinate of the current location of a cleaning robot. ML methods use data to learn predictors h such that $\hat{y} \approx y$ (with true label y).

In principle, ML methods could use any possible map $h : \mathcal{X} \to \mathcal{Y}$ to predict the label $y \in \mathcal{Y}$ via computing $\hat{y} = h(\mathbf{x})$. The set of all maps from the feature space $\mathcal{X}$ to the label space is typically denoted as $\mathcal{Y}^{\mathcal{X}}$.[1] In general, the set $\mathcal{Y}^{\mathcal{X}}$ is way too large to be searched over by a practical ML methods. As a point in case, consider data points characterized by a single numeric feature $x \in \mathbb{R}$ and label $y \in \mathbb{R}$. The set of all real-valued maps $h(x)$ of a real-valued argument already contains uncountably infinite many different hypothesis maps [56].

Practical ML methods can search and evaluate only a (tiny) subset of all possible hypothesis maps. This subset of computationally feasible ("affordable") hypothesis maps is referred to as the hypothesis space or model underlying a ML method. As depicted in Figure 2.11, ML methods typically use a hypothesis space $\mathcal{H}$ that is a tiny subset of $\mathcal{Y}^{\mathcal{X}}$. Similar to the

[1]The notation $\mathcal{Y}^{\mathcal{X}}$ is to be understood as a symbolic shorthand and should not be understood literately as a power such as 4^5.

feature x	prediction $h(x)$
0	0
1/10	10
2/10	3
$\vdots$	$\vdots$
1	22.3

Table 2.1: A look-up table defines a hypothesis map h. The value $h(x)$ is given by the entry in the second column of the row whose first column entry is x. We can construct a hypothesis space $\mathcal{H}$ by using a collection of different look-up tables.

features and labels used to characterize data points, also the hypothesis space underlying a ML method is a design choice. As we will see, the choice for the hypothesis space involves a trade-off between computational complexity and statistical properties of the resulting ML methods.

The preference for a particular hypothesis space often depends on the computational infrastructure that is available to a ML method. Different computational infrastructures favour different hypothesis spaces. ML methods implemented in a small embedded system, might prefer a linear hypothesis space which results in algorithms that require a small number of arithmetic operations. Deep learning methods implemented in a cloud computing environment typically use much larger hypothesis spaces obtained from large ANN (see Section 3.11).

ML methods can also be implemented using a spreadsheet software. Here, we might use a hypothesis space consisting of maps $h : \mathcal{X} \rightarrow \mathcal{Y}$ that are represented by look up tables (see Table 2.1). If we instead use the programming language Python to implement a ML method, we can obtain a hypothesis class by collecting all possible Python subroutines with one input (scalar feature x), one output argument (predicted label $\hat{y}$) and consisting of less than 100 lines of code.

Broadly speaking, the design choice for the hypothesis space $\mathcal{H}$ of a ML method has to balance between two conflicting requirements.

- It has to be sufficiently large such that it contains at least one accurate predictor map $\widehat{h} \in \mathcal{H}$. A hypothesis space $\mathcal{H}$ that is too small might fail to include a predictor map required to reproduce the (potentially highly non-linear) relation between features and label.

Consider the task of grouping or classifying images into "cat" images and "no cat image". The classification of each image is based solely on the feature vector obtained from the pixel colour intensities. The relation between features and label ($y \in \{\text{cat}, \text{no cat}\}$) is highly non-linear. Any ML method that uses a hypothesis space consisting only of linear maps will most likely fail to learn a good predictor (classifier). We say that a ML method is underfitting if it uses a hypothesis space that does not contain any hypotheses maps that can accurately predict the label of any data points.

- It has to be sufficiently small such that its processing fits the available computational resources (memory, bandwidth, processing time). We must be able to efficiently search over the hypothesis space to find good predictors (see Section 2.3 and Chapter 4). This requirement implies also that the maps $h(\mathbf{x})$ contained in $\mathcal{H}$ can be evaluated (computed) efficiently [4]. Another important reason for using a hypothesis space $\mathcal{H}$ that is not too large is to avoid overfitting (see Chapter 7). If the hypothesis space $\mathcal{H}$ is too large, then we can easily find a hypothesis which (almost) perfectly predicts the labels of data points in a training set which is used to learn a hypothesis. However, such a hypothesis might deliver poor predictions for labels of data points outside the training set. We say that the hypothesis does not generalize well.

2.2.1 Parametrized Hypothesis spaces

A wide range of current scientific computing environments allow for efficient numerical linear algebra. This hard- and software allows to efficiently process data that is provided in the form of numeric arrays such as vectors, matrices or tensors [112]. To take advantage of such computational infrastructure, many ML methods use the hypothesis space

$$\mathcal{H}^{(n)} := \{h^{(\mathbf{w})} : \mathbb{R}^n \to \mathbb{R} : h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{x}^T\mathbf{w} \text{ with some parameter vector } \mathbf{w} \in \mathbb{R}^n\}. \tag{2.4}$$

The hypothesis space (2.4) is constituted by linear maps (functions)

$$h^{(\mathbf{w})}\big(\mathbf{x}\big) : \mathbb{R}^n \to \mathbb{R} : \mathbf{x} \mapsto \mathbf{w}^T\mathbf{x}. \tag{2.5}$$

The function $h^{(\mathbf{w})}$ (2.5) maps, in a linear fashion, the feature vector $\mathbf{x} \in \mathbb{R}^n$ to the predicted label $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} \in \mathbb{R}$. For $n = 1$ the feature vector reduces a single feature x and the hypothesis space (2.4) consists of all maps $h^{(w)}(x) = wx$ with weight $w \in \mathbb{R}$ (see Figure 2.8).

The elements of the hypothesis space $\mathcal{H}$ in (2.4) are parametrized by the parameter

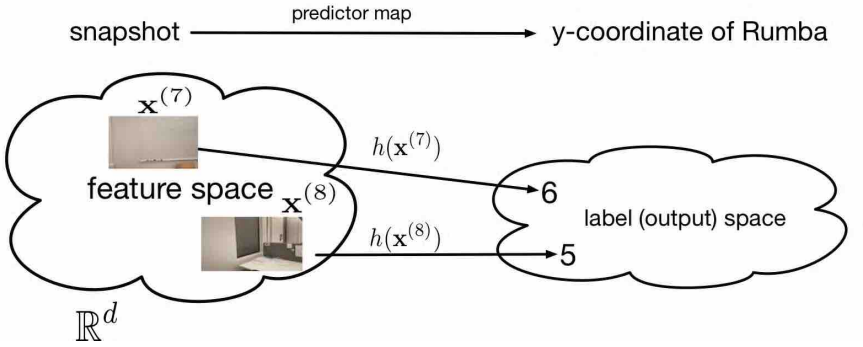


Figure 2.7: Consider a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that is used for locating a cleaning robot. The hypothesis h reads in the feature vector $\mathbf{x}^{(t)} \in \mathcal{X}$, that might be RGB pixel intensities of an on-board camera snapshot, at time t. It then outputs a prediction $\hat{y}^{(t)} = h(\mathbf{x}^{(t)})$ for the y-coordinate $y^{(t)}$ of the cleaning robot at time t. A key problem studied within ML is how to automatically learn a good (accurate) predictor h such that $y^{(t)} \approx h(\mathbf{x}^{(t)})$.

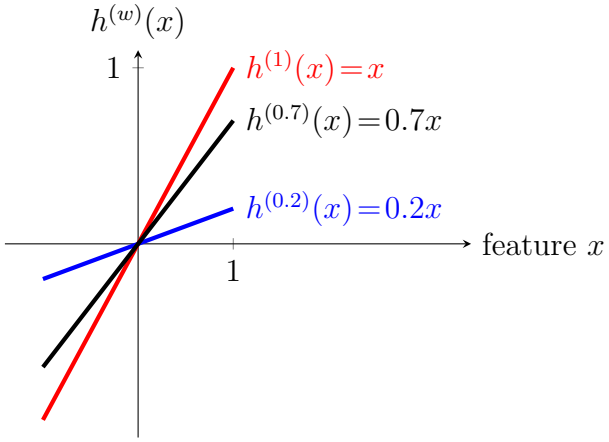


Figure 2.8: Three particular members of the hypothesis space $\mathcal{H} = \{h^{(w)} : \mathbb{R} \to \mathbb{R}, h^{(w)}(x) = wx\}$ which consists of all linear functions of the scalar feature x. We can parametrize this hypothesis space conveniently using the weight $w \in \mathbb{R}$ as $h^{(w)}(x) = wx$.

vector $\mathbf{w} \in \mathbb{R}^n$. Each map $h^{(\mathbf{w})} \in \mathcal{H}$ is fully specified by the parameter vector $\mathbf{w} \in \mathbb{R}^n$. This parametrization of the hypothesis space $\mathcal{H}$ allows to process and manipulate hypothesis maps by vector operations. In particular, instead of searching over the function space $\mathcal{H}$ (its elements are functions!) to find a good hypothesis, we can equivalently search over all possible parameter vectors $\mathbf{w} \in \mathbb{R}^n$.

The search space $\mathbb{R}^n$ is still (uncountably) infinite but it has a rich geometric and algebraic structure that allows to efficiently search over this space. Chapter 5 discusses methods that use the concept ot a gradient to implement an efficient search for useful parameter vectors $\mathbf{w} \in \mathbb{R}^n$.

The hypothesis space (2.4) is also appealing because of the broad availability of computing hardware such as graphic processing units. Another factor boosting the widespread use of (2.4) might be the offer for optimized software libraries for numerical linear algebra.

The hypothesis space (2.4) can also be used for classification problems, e.g., with label space $\mathcal{Y} = \{-1, 1\}$. Indeed, given a linear predictor map $h^{(\mathbf{w})}$ we can classify data points according to $\hat{y}=1$ for $h^{(\mathbf{w})}(\mathbf{x}) \geq 0$ and $\hat{y}=-1$ otherwise. We refer to a classifier that computes the predicted label by first applying a linear map to the features as a linear classifier.

Figure 2.9 illustrates the decision regions (2.3) of a linear classifier for binary labels. The decision regions are half-spaces and, in turn, the decision boundary is a hyperplane $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} = b\}$. Note that each linear classifier corresponds to a particular linear hypothesis map from the hypothesis space (2.4). However, we can use different loss functions to measure the quality of a linear classifier.

Three widely-used examples for ML methods that learn a linear classifier are logistic regression (see Section 3.6), the SVM (see Section 3.7) and the naive Bayes classifier (see Section 3.8).

In some application domains, the relation between features $\mathbf{x}$ and label y of a data point is highly non-linear. As a case in point, consider data points that are images of animals. The map that relates the pixel intensities of an image to a label value indicating if the image shows a cat is highly non-linear. For such applications, the hypothesis space (2.4) is not suitable as it only contains linear maps.

Another popular example for a parametrized hypothesis space studied in this book, besides the space of linear maps (2.4), contains also non-linear maps. This parametrized hypothesis space is obtained from a parametrized signal flow diagram which is referred to as an ANN. Section 3.11 discusses the construction of non-linear parametrized hypothesis spaces using an ANN.

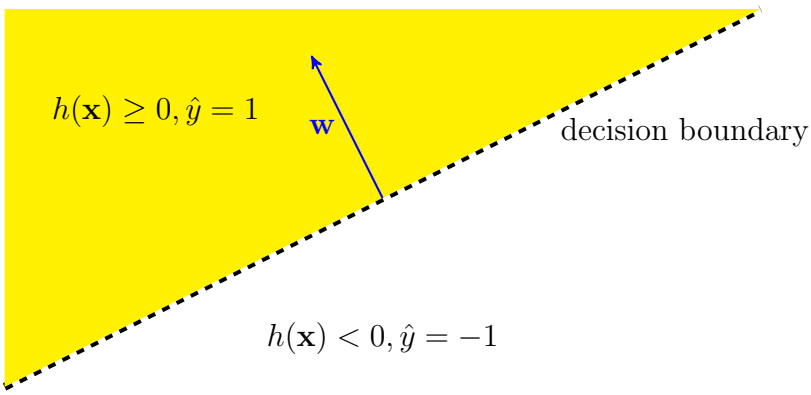


Figure 2.9: A hypothesis $h : \mathcal{X} \to \mathcal{Y}$ for a binary classification problem, with label space $\mathcal{Y} = \{-1, 1\}$ and feature space $\mathcal{X} = \mathbb{R}^2$, can be represented conveniently via the decision boundary (dashed line) which separates all feature vectors $\mathbf{x}$ with $h(\mathbf{x}) \geq 0$ from the region of feature vectors with $h(\mathbf{x}) < 0$. If the decision boundary is a hyperplane $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} = b\}$ (with normal vector $\mathbf{w} \in \mathbb{R}^n$), we refer to the map h as a linear classifier.

Upgrading a Hypothesis Space via Feature Maps. Let us discuss a simple but powerful technique for enlarging ("upgrading") a given hypothesis space $\mathcal{H}$ to a larger hypothesis space $\mathcal{H}' \supseteq \mathcal{H}$ that offers a wider selection of hypothesis maps. The idea is to replace the original features $\mathbf{x}$ of a data point with new (transformed) features $\mathbf{z} = \boldsymbol{\Phi}(\mathbf{x})$. The transformed features are obtained by applying a feature map $\phi(\cdot)$ to the original features $\mathbf{x}$. This upgraded hypothesis space $\mathcal{H}'$ consists of all concatenations of the feature map ϕ and some hypothesis $h \in \mathcal{H}$,

$$\mathcal{H}' := \big\{h'(\mathbf{x}) := h\big(\boldsymbol{\Phi}(\mathbf{x})\big) : h \in \mathcal{H}\big\}. \tag{2.6}$$

The construction (2.6) allows for different combinations of a feature map $\boldsymbol{\Phi}$ and a "base" hypothesis space $\mathcal{H}$. The only requirement is that the output of the feature map can be used as (is compatible with the) input of each hypothesis $h \in \mathcal{H}$. Mathematically speaking, the range of the feature map $\boldsymbol{\Phi}$ must belong to the domain of the maps in $\mathcal{H}$.

Examples for ML methods that use a hypothesis space of the form (2.6) include polynomial regression (see Section 3.2), Gaussian basis regression (see Section 3.5) and kernel methods (see Section 3.9). The feature map in (2.6) might also be obtained from clustering or feature learning methods (see Section 8.4 and Section 9.2.1). One important family of ML methods that learn the feature map in (2.6) is deep learning (see Section 3.11). These methods use ANNs consisting of several layers with the final layer being a linear map and

the previous layers implementing a feature map of the original features (see Figure 2.10).

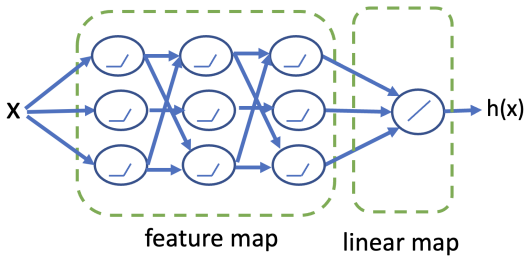


Figure 2.10: An ANN with several hidden layers that serve as a tunable (learnable) feature map which is concatenated with a linear map in the final (output) layer.

For the special case of the linear hypothesis space (2.4), the resulting enlarged hypothesis space (2.6) is given by all linear maps $\mathbf{w}^T\mathbf{z}$ of the transformed features $\mathbf{\Phi}(\mathbf{x})$. Combining the hypothesis space (2.4) with a non-linear feature map results in a hypothesis space that contains non-linear maps from the original features $\mathbf{x}$ to the predicted label $\hat{y}$,

$$\hat{y} = \mathbf{w}^T\mathbf{z} = \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}). \tag{2.7}$$

Non-Numeric Features. The hypothesis space (2.4) can only be used for data points whose features are numeric vectors $\mathbf{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$. In some application domains, such as natural language processing, there is no obvious natural choice for numeric features. However, since ML methods based on the hypothesis space (2.4) are well developed (using numerical linear algebra), it might be useful to construct numerical features even for non-numeric data (such as text). For text data, there has been significant progress recently on methods that map a human-generated text into sequences of vectors (see [48, Chap. 12] for more details). Moreover, Section 9.3 will discuss an approach to generate numeric features for data points that have an intrinsic notion of similarity.

2.2.2 The Size of a Hypothesis Space

The notion of a hypothesis space being too small or being too large can be made precise in different ways. The size of a finite hypothesis space $\mathcal{H}$ can be defined as its cardinality $|\mathcal{H}|$ which is simply the number of its elements. For example, consider data points represented by $100 \times 10 = 1000$ black-and-white pixels and characterized by a binary label $y \in \{0, 1\}$. We can model such data points using the feature space $\mathcal{X} = \{0, 1\}^{1000}$ and label space $\mathcal{Y} = \{0, 1\}$. The largest possible hypothesis space $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$ consists of all maps from $\mathcal{X}$ to $\mathcal{Y}$. The size

or cardinality of this space is $|\mathcal{H}| = 2^{2^{1000}}$.

Many ML methods use a hypothesis space which contains infinitely many different predictor maps (see, e.g., (2.4)). For an infinite hypothesis space, we cannot use the number of its elements as a measure for its size. Indeed, for an infinite hypothesis space, the number of elements is not well-defined. Therefore, we measure the size of a hypothesis space $\mathcal{H}$ using its effective dimension $d_{\text{eff}}(\mathcal{H})$.

Consider a hypothesis space $\mathcal{H}$ consisting of maps $h : \mathcal{X} \to \mathcal{Y}$ that read in the features $\mathbf{x} \in \mathcal{X}$ and output an predicted label $\hat{y} = h(\mathbf{x}) \in \mathcal{Y}$. We define the effective dimension $d_{\text{eff}}(\mathcal{H})$ of $\mathcal{H}$ as the maximum number $D \in \mathbb{N}$ such that for any set $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(D)}, y^{(D)}\big)\big\}$ of D data points with different features, we can always find a hypothesis $h \in \mathcal{H}$ that perfectly fits the labels, $y^{(i)} = h\big(\mathbf{x}^{(i)}\big)$ for $i = 1, \ldots, D$.

The effective dimension of a hypothesis space is closely related to the Vapnik–Chervonenkis (VC) dimension [145]. The VC dimension is maybe the most widely used concept for measuring the size of infinite hypothesis spaces [145, 126, 12, 58]. However, the precise definition of the VC dimension are beyond the scope of this book. Moreover, the effective dimension captures most of the relevant properties of the VC dimension for our purposes. For a precise definition of the VC dimension and discussion of its applications in ML we refer to [126]. Let us illustrate the concept of effective dimension as a measure for the size of a hypothesis space with two examples: linear regression and polynomial regression.

Linear regression uses the hypothesis space

$$\mathcal{H}^{(n)} = \{h : \mathbb{R}^n \to \mathbb{R} : h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \text{ with some vector } \mathbf{w} \in \mathbb{R}^n\}.$$

Consider a dataset $\mathcal{D}' = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$ consisting of m data points. We refer to this number also as the sample size of the dataset $\mathcal{D}'$. Each data point is characterized by a feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and a numeric label $y^{(i)} \in \mathbb{R}$.

Let us further assume that data points are realizations of i.i.d. RVs with a common probability distribution. Under this i.i.d. assumption, the feature matrix

$$\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big) \in \mathbb{R}^{n \times m},$$

which is obtained by stacking (column-wise) the feature vectors $\mathbf{x}^{(i)}$ (for $i = 1, \ldots, m$), is full (column-) rank with probability one as long as $m \leq n$. Basic linear algebra tells us that in this case we can find a linear map $h \in \mathcal{H}^{(n)}$ that perfectly fits the data points in $\mathcal{D}'$.

To summarize, as long as the number m of data points in $\mathcal{D}'$ does not exceed the number

of features characterizing each data point, i.e., as long as $m \leq n$, we can find (with probability one) a parameter vector $\widehat{\mathbf{w}}$ such that $y^{(i)} = \widehat{\mathbf{w}}^T \mathbf{x}^{(i)}$ for all $i = 1, \ldots, m$. Conversely, for $m > n$ it is easy to verify that with probability one, a dataset $\mathcal{D}'$ consisting of m (i.i.d.) data points cannot be perfectly fit by any linear hypothesis. Thus, the effective dimension of the linear hypothesis space $\mathcal{H}^{(n)}$ is $D = n$.

As a second example, consider the hypothesis space $\mathcal{H}^{(n)}_{\text{poly}}$ which is constituted by the set of polynomials with maximum degree n. The fundamental theorem of algebra tells us that any set of m data points with different features can be perfectly fit by a polynomial of degree n as long as $n \geq m$. Therefore, the effective dimension of the hypothesis space $\mathcal{H}^{(n)}_{\text{poly}}$ is $D = n$. Section 3.2 discusses polynomial regression in more detail.

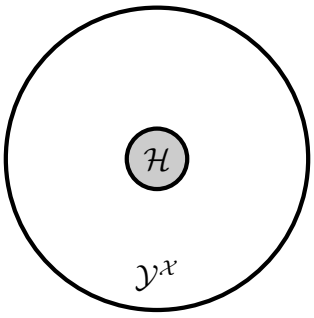


Figure 2.11: The hypothesis space $\mathcal{H}$ is a (typically very small) subset of the (typically very large) set $\mathcal{Y}^{\mathcal{X}}$ of all possible maps from feature space $\mathcal{X}$ into the label space $\mathcal{Y}$.

2.3 The Loss

Every ML method uses a (more of less explicit) hypothesis space $\mathcal{H}$ which consists of all computationally feasible predictor maps h. Which predictor map h out of all the maps in the hypothesis space $\mathcal{H}$ is the best for the ML problem at hand? To answer this questions, ML methods use the concept of a loss function. Formally, a loss function is a map

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

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

The loss value $L\left((\mathbf{x}, y), h\right)$ quantifies the discrepancy between the true label y and the

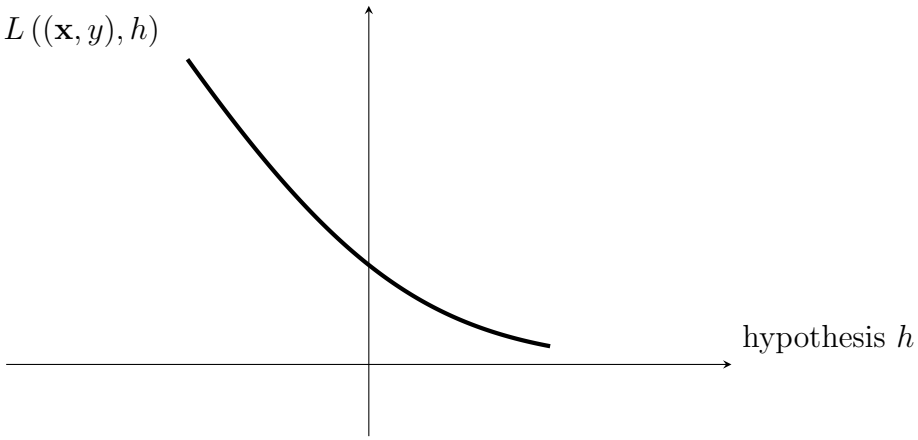


Figure 2.12: Some loss function $L\left((\mathbf{x}, y), h\right)$ for a fixed data point, with features $\mathbf{x}$ and label y, and varying hypothesis h. ML methods try to find (learn) a hypothesis that incurs minimum loss.

predicted label $h(\mathbf{x})$. A small (close to zero) value $L\left((\mathbf{x}, y), h\right)$ indicates a low 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 $h \in \mathcal{H}$. The basic principle of ML methods can then be formulated as: Learn (find) a hypothesis out of a given hypothesis space $\mathcal{H}$ that incurs a minimum loss $L\left((\mathbf{x}, y), h\right)$ for any data point (see Chapter 4).

Much like the choice for the hypothesis space $\mathcal{H}$ used in a ML method, also the loss function is a design choice. We will discuss some widely used examples for loss function in Section 2.3.1 and Section 2.3.2. The choice for the loss function should take into account the computational complexity of searching the hypothesis space for a hypothesis with minimum loss. Consider a ML method that uses a parametrized hypothesis space and a loss function that is a convex and differentiable function of the parameters of a hypothesis. In this case, we can efficiently search for a hypothesis with small loss using the gradient-based methods discussed in Chapter 5. The minimization of a loss function that is either non-convex or non-differentiable tends to be computationally much more difficult. Section 4.2 discusses the computational complexities of different types of loss functions in more detail.

Beside computational aspects, the choice for the loss function should also take into account statistical aspects. For example, some loss functions result in ML methods that are more robust against outliers (see Section 3.3 and Section 3.7). The choice of loss function might also be guided by probabilistic models for the data generated in an ML application. Section 3.12 details how the maximum likelihood principle of statistical inference provides an explicit construction of loss functions in terms of an (assumed) probability distribution

for data points.

The choice for the loss function used to evaluate the quality of a hypothesis might also be influenced by its interpretability. Section 2.3.2 discusses loss functions for hypotheses that are used to classify data points into two categories. It seems natural to measure the quality of such a hypothesis by the average number of wrongly classified data points, which is precisely the average 0/1 loss (2.9) (see Section 2.3.2).

In contrast to its appealing interpretation as error-rate, the computational aspects of the average 0/1 loss are less pleasant. Minimizing the average 0/1 loss to learn an accurate hypothesis amounts to a non-convex and non-smooth optimization problem which is computationally challenging. Section 2.3.2 introduces the logistic loss as a computationally attractive alternative choice for the loss function in binary classification problems. Learning a hypothesis that minimizes a (average) logistic loss amounts to a smooth convex optimization problem. Chapter 5 discusses computationally cheap gradient-based methods for solving smooth convex optimization problems.

The above aspects (computation, statistic, interpretability) result typically in conflicting goals for the choice of a loss function. A loss function that has favourable statistical properties might incur a high computational complexity of the resulting ML method. Loss functions that result in computationally efficient ML methods might not allow for an easy interpretation (what does it mean intuitively if the logistic loss of a hypothesis in a binary classification problem is 10^{-1}?). It might therefore be useful to use different loss functions for the search of a good hypothesis (see Chapter 4) and for its final evaluation. Figure 2.13 depicts an example for two such loss functions, one of them used for learning a hypothesis by minimizing the loss and the other one used for the final performance evaluation.

For example, in a binary classification problem, we might use the logistic loss to search for (learn) an accurate hypothesis using the optimization methods in Chapter 4. The logistic loss is appealing for this purpose as it can be minimized via efficient gradient-based methods (see Chapter 5). After having found (learnt) an accurate hypothesis, we use the average 0/1 loss for the final performance evaluation. The 0/1 loss is appealing for this purpose as it can be interpreted as an error or misclassification rate. The loss function used for the final performance evaluation of a learnt hypothesis is sometimes referred to as metric.

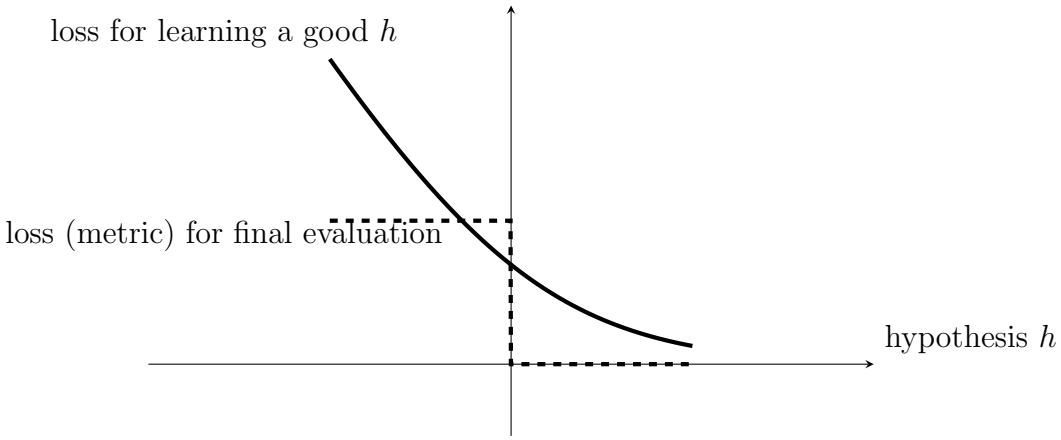


Figure 2.13: Two different loss functions for a given data point and varying hypothesis h. One of these loss functions (solid curve) is used to learn a good hypothesis by minimizing the loss. The other loss function (dashed curve) is used to evaluate the performance of the learnt hypothesis. The loss function used for this final performance evaluation is sometimes referred to as a metric.

2.3.1 Loss Functions for Numeric Labels

For ML problems involving data points with a numeric label $y \in \mathbb{R}$, i.e., for regression problems (see Section 2.1.2), a widely used (first) choice for the loss function can be the squared error loss

$$L\left((\mathbf{x}, y), h\right) := \big(y - \underbrace{h(\mathbf{x})}_{=\hat{y}}\big)^2. \tag{2.8}$$

The squared error loss (2.8) depends on the features $\mathbf{x}$ only via the predicted label value $\hat{y} = h(\mathbf{x})$. We can evaluate the squared error loss solely using the prediction $h(\mathbf{x})$ and the true label value y. Besides the prediction $h(\mathbf{x})$, no other properties of the features $\mathbf{x}$ are required to determine the squared error loss. We will slightly abuse notation and use the shorthand $L\left(y, \hat{y}\right)$ for any loss function that depends on the features $\mathbf{x}$ only via the predicted label $\hat{y} = h(\mathbf{x})$. Figure 2.14 depicts the squared error loss as a function of the prediction error $y - \hat{y}$.

The squared error loss (2.8) has appealing computational and statistical properties. For linear predictor maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the squared error loss is a convex and differentiable function of the weight vector $\mathbf{w}$. This allows, in turn, to efficiently search for the optimal linear predictor using efficient iterative optimization methods (see Chapter 5). The squared error loss also has a useful interpretation in terms of a probabilistic model for the features and labels. Minimizing the squared error loss is equivalent to maximum likelihood estimation

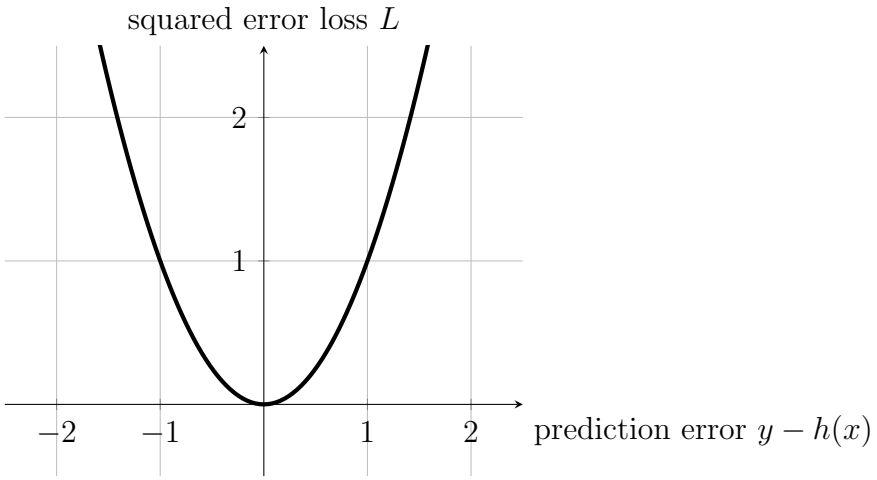


Figure 2.14: A widely used choice for the loss function in regression problems (with data points having numeric labels) is the squared error loss (2.8). Note that, for a given hypothesis h, we can evaluate the squared error loss only if we know the features $\mathbf{x}$ and the label y of the data point.

within a linear Gaussian model [58, Sec. 2.6.3].

Another loss function used in regression problems is the absolute error loss $|\hat{y} - y|$. Using this loss function to guide the learning of a predictor results in methods that are robust against a few outliers in the training set (see Section 3.3). However, this improved robustness comes at the expense of increased computational complexity of minimizing the (non-differentiable) absolute error loss compared to the (differentiable) squared error loss (2.8).

2.3.2 Loss Functions for Categorical Labels

Classification problems involve data points whose labels take on values from a discrete label space $\mathcal{Y}$. In what follows, unless stated otherwise, we focus on binary classification problems. Moreover, without loss of generality, we use the label space $\mathcal{Y} = \{-1, 1\}$. Classification methods aim at learning a hypothesis or classifier that maps the features $\mathbf{x}$ of a data point to a predicted label $\hat{y} \in \mathcal{Y}$.

It is often convenient to implement a classifier by thresholding the value $h(\mathbf{x}) \in \mathbb{R}$ of a hypothesis map that can deliver arbitrary real numbers. We then classify a data point as $\hat{y} = 1$ if $h(\mathbf{x}) > 0$ and $\hat{y} = -1$ otherwise. Thus, the predicted label is obtained from the sign of the value $h(\mathbf{x})$. While the sign of $h(\mathbf{x})$ determines the predicted label $\hat{y}$, we can interpret the absolute value $|h(\mathbf{x})|$ as the confidence in this classification. Is is customary to

abuse notation and refer to both, the final classification rule (obtained by a threshold step) $\mathbf{x} \mapsto \hat{y}$ and the hypothesis $h(\mathbf{x})$ (whose output is compared against a threshold) as a binary classifier.

In principle, we can measure the quality of a hypothesis when used to classify data points using the squared error loss (2.8). However, the squared error is typically a poor measure for the quality of a hypothesis $h(\mathbf{x})$ that is used to classify a data point with binary label $y \in \{-1, 1\}$. Figure 2.15 illustrates how the squared error loss of a hypothesis can be (highly) misleading for binary classification.

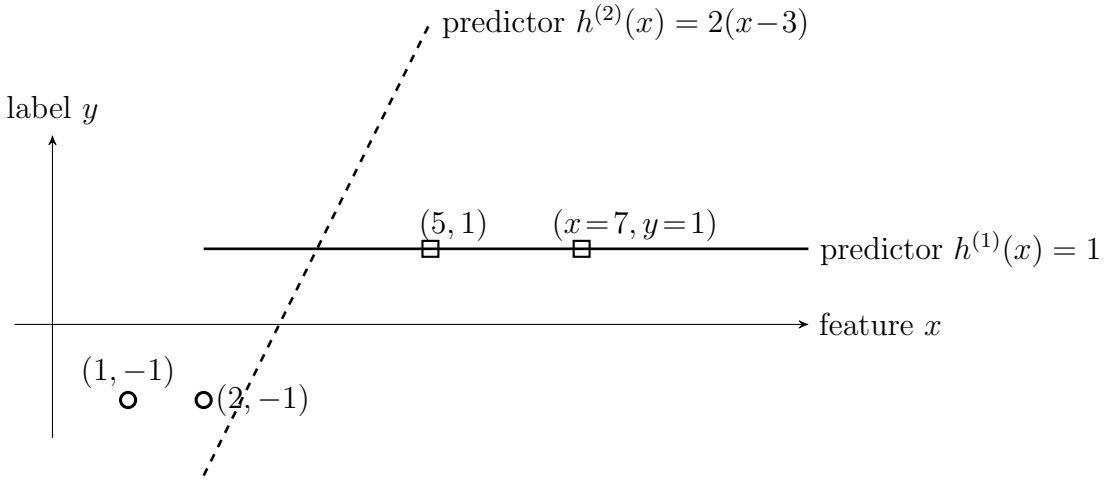


Figure 2.15: A training set consisting of four data points with binary labels $\hat{y}^{(i)} \in \{-1, 1\}$. Minimizing the squared error loss (2.8) would prefer the (poor) classifier $h^{(1)}$ over the (reasonable) classifier $h^{(2)}$.

Figure 2.15 depicts a dataset $\mathcal{D}$ consisting of $m = 4$ data points with binary labels $y^{(i)} \in \{-1, 1\}$, for $i = 1, \ldots, m$. The figure also depicts two candidate hypothesis maps $h^{(1)}(x)$ and $h^{(2)}(x)$ that can be used for classifying data points. The classifications $\hat{y}$ obtained with the hypothesis $h^{(2)}(x)$ would perfectly match the labels of the four training data points since $h^{(2)}\big(x^{(i)}\big) \geq 0$ if and if only if $y^{(i)} = 1$. In contrast, the classifications $\hat{y}^{(i)}$ obtained by thresholding $h^{(1)}(x)$ are wrong for data points with $y = -1$.

Looking at $\mathcal{D}$, it seems reasonable to prefer $h^{(2)}$ over $h^{(1)}$ for classifying data points. However, the squared error loss incurred by the (reasonable) classifier $h^{(2)}$ is much higher than the squared error loss incurred by the (poor) classifier $h^{(1)}$. The squared error loss is typically a bad choice for assessing the quality of a hypothesis that is used for classifying data points into different categories.

Generally speaking, we want the loss function to punish (deliver large values for) a hypothesis that is very confident ($|h(\mathbf{x})|$ is large) in a wrong classification ($\hat{y} \neq y$). Moreover, a useful loss function should not punish (deliver small values for) a hypothesis is very confident ($|h(\mathbf{x})|$ is large) in a correct classification ($\hat{y} = y$). However, by its very definition, the squared error loss (2.8) yields large values if the confidence $|h(\mathbf{x})|$ is large, no matter if the resulting classification (obtained after thresholding) is correct or wrong.

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

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

$$L\left((\mathbf{x}, y), h\right) := \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{else,} \end{cases} \text{ with } \hat{y} = 1 \text{ for } h(\mathbf{x}) \geq 0, \text{ and } \hat{y} = -1 \text{ for } h(\mathbf{x}) < 0. \quad (2.9)$$

Figure 2.16 illustrates the 0/1 loss (2.9) for a data point with features $\mathbf{x}$ and label $y{=}1$ as a function of the hypothesis value $h(\mathbf{x})$. The 0/1 loss is equal to zero if the hypothesis yields a correct classification $\hat{y} = y$. For a wrong classification $\hat{y} \neq y$, the 0/1 loss yields the value one.

The 0/1 loss (2.9) is conceptually appealing when data points are interpreted as realizations of i.i.d. RVs with the same probability distribution $p(\mathbf{x}, y)$. Given m realizations $(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ of such i.i.d. RVs,

$$(1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right) \approx p(y \neq \hat{y}) \quad (2.10)$$

with high probability for sufficiently large sample size m. A precise formulation of the approximation (2.10) can be obtained from the law of large numbers [11, Section 1]. We can apply the law of large numbers since the loss values $L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right)$ are realizations of i.i.d. RVs. It is customary to indicate the average 0/1 loss of a hypothesis indirectly via the accuracy $1 - (1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right)$.

In view of (2.10), the 0/1 loss (2.9) seems a very natural choice for assessing the quality of a classifier if our goal is to enforce correct classifications $\hat{y} = y$. This appealing statistical property of the 0/1 loss comes at the cost of a high computational complexity. Indeed, for a given data point $(\mathbf{x}, y)$, the 0/1 loss (2.9) is non-convex and non-differentiable when viewed as a function of the hypothesis h. Thus, ML methods that use the 0/1 loss to learn a hypothesis map typically involve advanced optimization methods to solve the resulting learning problem (see Section 3.8).

To avoid the non-convexity of the 0/1 loss (2.9) we might approximate it by a convex loss function. One popular convex approximation of the 0/1 loss is the hinge loss

$$L\left((\mathbf{x}, y), h\right) := \max\{0, 1 - yh(\mathbf{x})\}. \tag{2.11}$$

Figure 2.16 depicts the hinge loss (2.11) as a function of the hypothesis $h(\mathbf{x})$. The hinge loss (2.11) becomes minimal (equal to zero) for a correct classification ($\hat{y} = y$) with sufficient confidence $h(\mathbf{x}) \geq 1$. For a wrong classification ($\hat{y} \neq y$), the hinge loss increases monotonically with the confidence $|h(x)|$ in the wrong classification. While the hinge loss avoids the non-convexity of the 0/1 loss, it still is a non-differentiable function of $h(\mathbf{x})$. A non-differentiable loss function cannot be minimized by simple gradient-based methods (see Chapter 5) but require more advanced optimization methods.

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

$$L\left((\mathbf{x}, y), h\right) := \log(1 + \exp(-yh(\mathbf{x}))). \tag{2.12}$$

The logistic loss (2.12) is used in logistic regression (see Section 3.6) to measure the usefulness of a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Figure 2.16 depicts the logistic loss (2.12) as a function of the hypothesis $h(\mathbf{x})$. The logistic loss (2.12) is a convex and differentiable function of $h(\mathbf{x})$. For a correct classification ($\hat{y} = y$), the logistic loss (2.12) decreases monotonically with increasing confidence $h(\mathbf{x})$. For a wrong classification ($\hat{y} \neq y$), the logistic loss increases monotonically with increasing confidence $|h(\mathbf{x})|$ in the wrong classification.

Both the hinge loss (2.11) and the logistic loss (2.12) are convex functions of the weights $\mathbf{w} \in \mathbb{R}^n$ in a linear hypothesis map $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. However, in contrast to the hinge loss, the logistic loss (2.12) is also a differentiable function of the $\mathbf{w}$. The convex and differentiable logistic loss function can be minimized using simple gradient-based methods such as gradient descent (GD) (see Chapter 5.5). In contrast, we cannot use basic gradient-based methods

to minimize the hinge loss since it does not have a gradient for $h(\mathbf{x})y = 1$. However, we can apply a generalization of GD which is known as subgradient descent [16]. Subgradient descent is obtained from GD by generalizing the concept of a gradient to that of a subgradient.

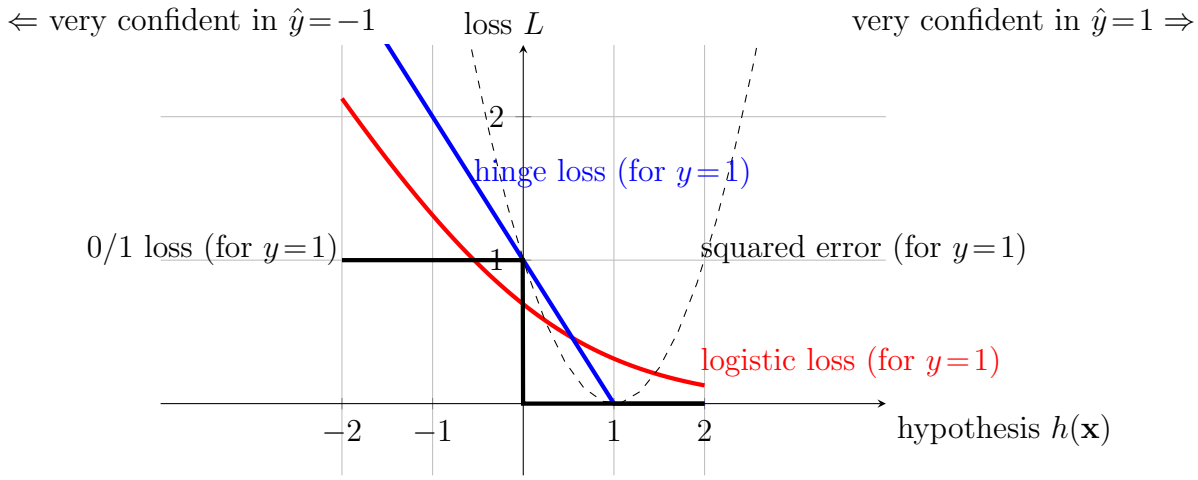


Figure 2.16: The solid curves depict three widely-used loss functions for binary classification. A data point with features $\mathbf{x}$ and label $y = 1$ is classified as $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$ and classified as $\hat{y} = -1$ if $h(\mathbf{x}) < 0$. We can interpret the absolute value $|h(\mathbf{x})|$ as the confidence in the classification $\hat{y}$. The more confident we are in a correct classification ($\hat{y} = y = 1$), i.e, the more positive $h(\mathbf{x})$, the smaller the loss. Note that each of the three loss functions for binary classification tends monotonically towards 0 for increasing $h(\mathbf{x})$. The dashed curve depicts the squared error loss (2.8), which increases for increasing $h(\mathbf{x})$.

2.3.3 Loss Functions for Ordinal Label Values

Some loss functions are particularly suited for predicting ordinal label values (see Section 2.1). Consider data points representing areal images of rectangular areas of size 1km by 1km. We characterize each data point (rectangular area) by the feature vector $\mathbf{x}$ obtained by stacking the RGB values of each image pixel (see Figure 2.5). Beside the feature vector, each rectangular area is characterized by a label $y \in \{1, 2, 3\}$ where

- $y = 1$ means that the area contains no trees.
- $y = 2$ means that the area is partially covered by trees.
- $y = 3$ means that the are is entirely covered by trees.

We might consider the label value $y = 2$ to be "larger" than label value $y = 1$ and label value $y = 3$ to be "larger" than label value $y = 2$. Let us construct a loss function that takes

such an ordering of label values into account when evaluating the quality of the predictions $h(\mathbf{x})$.

Consider a data point with feature vector $\mathbf{x}$ and label $y = 1$ as well as two different hypotheses $h^{(a)}, h^{(b)} \in \mathcal{H}$. The hypothesis $h^{(a)}$ delivers the predicted label $\hat{y}^{(a)} = h^{(a)}(\mathbf{x}) = 2$, while the other hypothesis $h^{(b)}$ delivers the predicted label $\hat{y}^{(a)} = h^{(a)}(\mathbf{x}) = 3$. Both predictions are wrong, since they are different form the true label value $y = 1$. It seems reasonable to consider the prediction $\hat{y}^{(a)}$ to be less wrong than the prediction $\hat{y}^{(b)}$ and therefore we would prefer the hypothesis $h^{(a)}$ over $h^{(b)}$. However, the 0/1 loss is the same for $h^{(a)}$ and $h^{(b)}$ and therefore does not reflect our preference for $h^{(a)}$. We need to modify (or tailor) the 0/1 loss to take into account the application-specific ordering of label values. For the above application, we might define a loss function via

$$L\left((\mathbf{x}, y), h\right) := \begin{cases} 0 & \text{, when } y = h(\mathbf{x}) \\ 10 & \text{, when } |y - h(\mathbf{x})| = 1 \\ 100 & \text{otherwise.} \end{cases} \tag{2.13}$$

2.3.4 Empirical Risk

The basic idea of ML methods (including those discussed in Chapter 3) is to find (or learn) a hypothesis (out of a given hypothesis space $\mathcal{H}$) that incurs minimum loss when applied to arbitrary data points. To make this informal goal precise we need to specify what we mean by "arbitrary data point". One of the most successful approaches to define the notion of "arbitrary data point" is by probabilistic models for the observed data points.

The most basic and widely-used probabilistic model interprets data points $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ as realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{x}, y)$. Given such a probabilistic model, it seems natural to measure the quality of a hypothesis h by the expected loss or Bayes risk [85]

$$\mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\} := \int_{\mathbf{x},y} L\left((\mathbf{x}, y), h\right) dp(\mathbf{x}, y). \tag{2.14}$$

The Bayes risk of h is the expected value of the loss $L\left((\mathbf{x}, y), h\right)$ incurred when applying the hypothesis h to (the realization of) a random data point with features $\mathbf{x}$ and label y. Note that the computation of the Bayes risk (2.15) requires the joint probability distribution $p(\mathbf{x}, y)$ of the (random) features and label of data points.

The Bayes risk (2.15) seems to be reasonable performance measure for a hypothesis h. Indeed, the Bayes risk of a hypothesis is small only if the hypothesis incurs a small loss on average for data points drawn from the probability distribution $p(\mathbf{x}, y)$. However, it might be challenging to verify if the data points generated in a particular application domain can be accurately modelled as realizations (draws) from a probability distribution $p(\mathbf{x}, y)$. Moreover, it is also often the case that we do not know the correct probability distribution $p(\mathbf{x}, y)$.

Let us assume for the moment, that data points are generated as i.i.d. realizations of a common probability distribution $p(\mathbf{x}, y)$ which is known. It seems reasonable to learn a hypothesis h^* that incurs minimum Bayes risk,

$$\mathbb{E}\big\{L\left((\mathbf{x}, y), h^*\right)\big\} := \min_{h \in \mathcal{H}} \mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\}. \tag{2.15}$$

A hypothesis that solves (2.15), i.e., that achieves the minimum possible Bayes risk, is referred to as a Bayes estimator [85, Chapter 4]. The main computational challenge for learning the optimal hypothesis is the efficient (numerical) solution of the optimization problem (2.15). Efficient methods to solve the optimization problem (2.15) are studied within estimation theory [85, 153].

The focus of this book is on ML methods which do not require knowledge of the underlying probability distribution $p(\mathbf{x}, y)$. One of the most widely used principle for these ML methods is to approximate the Bayes risk by an empirical (sample) average over a finite set of labeled data points $\mathcal{D} = \big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)$. In particular, we define the empirical risk of a hypothesis $h \in \mathcal{H}$ for a dataset $\mathcal{D}$ as

$$\widehat{L}(h|\mathcal{D}) = (1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right). \tag{2.16}$$

The empirical risk of the hypothesis $h \in \mathcal{H}$ is the average loss on the data points in $\mathcal{D}$. To ease notational burden, we use $\widehat{L}(h)$ as a shorthand for $\widehat{L}(h|\mathcal{D})$ if the underlying dataset $\mathcal{D}$ is clear from the context. Note that in general the empirical risk depends on both, the hypothesis h and the (features and labels of the) data points in the dataset $\mathcal{D}$.

If the data points used to compute the empirical risk (2.16) are (can be modelled as) realizations of i.i.d. RVs whose common distribution is $p(\mathbf{x}, y)$, basic results of probability theory tell us that

$$\mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\} \approx (1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right) \text{ for sufficiently large sample size } m. \tag{2.17}$$

The approximation error in (2.17) can be quantified precisely by some of the most basic results of probability theory. These results are often summarized under the umbrella term law of large numbers [11, 50, 9].

Many (if not most) ML methods are motivated by (2.17) which suggests that a hypothesis with small empirical risk (2.16) will also result in a small expected loss. The minimum possible expected loss is achieved by the Bayes estimator of the label y, given the features $\mathbf{x}$. However, to actually compute the optimal estimator we would need to know the (joint) probability distribution $p(\mathbf{x}, y)$ of features $\mathbf{x}$ and label y.

Confusion Matrix

Consider a dataset $\mathcal{D}$ with data points characterized by feature vectors $\mathbf{x}^{(i)}$ and labels $y^{(i)} \in \{1, \ldots, k\}$. We might interpret the label value of a data point as the index of a category or class to which the data point belongs to. Multi-class classification problems aim at learning a hypothesis h such that $h(\mathbf{x}) \approx y$ for any data point.

In principle, we could measure the quality of a given hypothesis h by the average 0/1 loss incurred on the labeled data points in (the training set) $\mathcal{D}$. However, if the dataset $\mathcal{D}$ contains mostly data points with one specific label value, the average 0/1 loss might obscure the performance of h for data points having one of the rare label values. Indeed, even if the average 0/1 loss is very small, the hypothesis might perform poorly for data points of a minority category.

The confusion matrix generalizes the concept of the 0/1 loss to application domains where the relative frequency (fraction) of data points with a specific label value varies significantly (imbalanced data). Instead of considering only the average 0/1 loss incurred by a hypothesis on a dataset $\mathcal{D}$, we use a whole family of loss functions. In particular, for each pair of two different label values $c, c' \in \{1, \ldots, k\}$, $c \neq c'$, we define the loss function

$$L^{(c \to c')}\big((\mathbf{x}, y), h\big) := \begin{cases} 1 & \text{if } y = c \text{ and } h(\mathbf{x}) = c' \\ 0 & \text{otherwise} \end{cases}. \tag{2.18}$$

We then compute the average loss (2.18) incurred on the dataset $\mathcal{D}$,

$$\widehat{L}^{(c \to c')}(h|\mathcal{D}) := (1/m) \sum_{i=1}^{m} L^{(c \to c')}\big((\mathbf{x}^{(i)}, y^{(i)}), h\big) \text{ for } c, c' \in \{1, \ldots, k\}. \tag{2.19}$$

It is convenient to arrange the values (2.19) as a matrix which is referred to as a confusion

matrix. The rows of a confusion matrix correspond to different values c of the true label of a data point. The columns of a confusion matrix correspond to different values c' delivered by the hypothesis $h(\mathbf{x})$. The (c, c')-th entry of the confusion matrix is $\widehat{L}^{(c \to c')}(h|\mathcal{D})$.

Precision, Recall and F-Measure

Consider an object detection application where data points are images. The label of data points might indicate the presence ($y = 1$) or absence ($y = -1$) of an object, it is then customary to define the [5]

$$\text{recall } := \widehat{L}^{(1\to 1)}(h|\mathcal{D}), \text{ and the precision } := \frac{\widehat{L}^{(1\to 1)}(h|\mathcal{D})}{\widehat{L}^{(1\to 1)}(h|\mathcal{D}) + \widehat{L}^{(-1\to 1)}(h|\mathcal{D})}. \tag{2.20}$$

Clearly, we would like to find a hypothesis with both, large recall and large precision. However, these two goals are typically conflicting, a hypothesis with a high recall will have small precision. Depending on the application, we might prefer having a high recall and tolerate a lower precision.

It might be convenient to combine the recall and precision of a hypothesis into a single quantity,

$$F_1 := 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.21}$$

The F measure (2.21) is the harmonic mean [2] of the precision and recall of a hypothesis h. It is a special case of the F_β-score

$$F_\beta := \big(1 + \beta^2\big)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}. \tag{2.22}$$

The F measure (2.21) is obtained from (2.22) for the choice $\beta = 1$. It is therefore customary to refer to (2.21) as the F_1-score of a hypothesis h.

2.3.5 Regret

In some ML applications, we might have access to the predictions obtained from some reference methods which are referred to as experts [20, 60]. The quality of a hypothesis h is measured via the difference between the loss incurred by its predictions $h(\mathbf{x})$ and the loss incurred by the predictions of the experts. This difference, which is referred to as the regret, measures by how much we regret to have used the prediction $h(\mathbf{x})$ instead of

using(or following) the prediction of the expert. The goal of regret minimization is to learn a hypothesis with a small regret compared to given set of experts.

The concept of regret minimization is useful when we do not make any probabilistic assumptions (see Section 2.1.4) about the data. Without a probabilistic model we cannot use the Bayes risk of the (optimal) Bayes estimator as a baseline (or benchmark). The concept of regret minimization avoids the need for a probabilistic model of the data to obtain a baseline [20]. This approach replaces the Bayes risk with the regret relative to given reference methods (the experts).

2.3.6 Rewards as Partial Feedback

Some applications involve data points whose labels are so difficult or costly to determine that we cannot assume to have any labeled data available. Without any labeled data, we cannot evaluate the loss function for different choices for the hypothesis. Indeed, the evaluation of the loss function typically amounts to measuring the distance between predicted label and true label of a data point. Instead of evaluating a loss function, we must rely on some indirect feedback or "reward" that indicates the usefulness of a particular prediction [20, 137].

Consider the ML problem of predicting the optimal steering directions for an autonomous car. The prediction has to be recalculated for each new state of the car. ML methods can sense the state via a feature vector $\mathbf{x}$ whose entries are pixel intensities of a snapshot. The goal is to learn a hypothesis map from the feature vector $\mathbf{x}$ to a guess $\hat{y} = h(\mathbf{x})$ for the optimal steering direction y (true label). Unless the car circles around in small area with fixed obstacles, we have no access to labeled data points or reference driving scenes for which we already know the optimum steering direction. Instead, the car (control unit) needs to learn the hypothesis $h(\mathbf{x})$ based solely on the feedback signals obtained from various sensing devices (cameras, distance sensors).

2.4 Putting Together the Pieces

A guiding theme of this book is that ML methods are obtained by different combinations of data, model and loss. We will discuss some key principles behind these methods in depth in the following chapters. Let us develop some intuition for how ML methods operate by considering a very simple ML problem. This problem involves data points that are characterized by a single numeric feature $x \in \mathbb{R}$ and a numeric label $y \in \mathbb{R}$. We assume to

have access to m labeled data points

$$\big(x^{(1)}, y^{(1)}\big), \ldots, \big(x^{(m)}, y^{(m)}\big) \tag{2.23}$$

for which we know the true label values $y^{(i)}$.

The assumption of knowing the exact true label values $y^{(i)}$ for any data point is an idealization. We might often face labelling or measurement errors such that the observed labels are noisy versions of the true label. Later on, we will discuss techniques that allow ML methods to cope with noisy labels in Chapter 7.

Our goal is to learn a (hypothesis) map $h : \mathbb{R} \to \mathbb{R}$ such that $h(x) \approx y$ for any data point. Given a data point with feature x, the function value $h(x)$ should be an accurate approximation of its label value y. We require the map to belong to the hypothesis space $\mathcal{H}$ of linear maps,

$$h^{(w_0,w_1)}(x) = w_1 x + w_0. \tag{2.24}$$

The predictor (2.24) is parameterized by the slope w_1 and the intercept (bias or offset) w_0. We indicate this by the notation $h^{(w_0,w_1)}$. A particular choice for the weights w_1, w_0 defines a linear hypothesis $h^{(w_0,w_1)}(x) = w_1 x + w_0$.

Let us use the linear hypothesis map $h^{(w_0,w_1)}(x)$ to predict the labels of training data points. In general, the predictions $\hat{y}^{(i)} = h^{(w_0,w_1)}\big(x^{(i)}\big)$ will not be perfect and incur a non-zero prediction error $\hat{y}^{(i)} - y^{(i)}$ (see Figure 2.17).

We measure the goodness of the predictor map $h^{(w_0,w_1)}$ using the average squared error loss (see (2.8))

$$\begin{aligned} f(w_0, w_1) &:= (1/m) \sum_{i=1}^{m} \big(y^{(i)} - h^{(w_0,w_1)}(x^{(i)})\big)^2 \\ &\overset{(2.24)}{=} (1/m) \sum_{i=1}^{m} \big(y^{(i)} - (w_1 x^{(i)} + w_0)\big)^2. \end{aligned} \tag{2.25}$$

The training error $f(w_0, w_1)$ is the average of the squared prediction errors incurred by the predictor $h^{(w_0,w_1)}(x)$ to the labeled data points (2.23).

It seems natural to learn a good predictor (2.24) by choosing the parameters w_0, w_1 to minimize the training error

$$\min_{w_0,w_1 \in \mathbb{R}} f(w_0, w_1) \overset{(2.25)}{=} \min_{w_1,w_0 \in \mathbb{R}} (1/m) \sum_{i=1}^{m} \big(y^{(i)} - (w_1 x^{(i)} + w_0)\big)^2. \tag{2.26}$$

The optimal parameters $\hat{w}_0, \hat{w}_1$ are characterized by the zero-gradient condition,[2]

$$\left.\frac{\partial f(w_0, w_1)}{\partial w_0}\right|_{w_0=\hat{w}_0, w_1=\hat{w}_1} = 0, \text{ and } \left.\frac{\partial f(w_0, w_1)}{\partial w_1}\right|_{w_0=\hat{w}_0, w_1=\hat{w}_1} = 0. \tag{2.27}$$

Inserting (2.25) into (2.27) and by using basic rules for calculating derivatives (see, e.g., [119]), we obtain the following optimality conditions

$$\begin{aligned} (1/m)\sum_{i=1}^{m}\big(y^{(i)} - (\hat{w}_1 x^{(i)} + \hat{w}_0)\big) = 0, \text{ and} \\ (1/m)\sum_{i=1}^{m} x^{(i)}\big(y^{(i)} - (\hat{w}_1 x^{(i)} + \hat{w}_0)\big) = 0. \end{aligned} \tag{2.28}$$

Any parameter values $\hat{w}_0, \hat{w}_1 \in \mathbb{R}$ that satisfy (2.28) define a hypothesis map $h^{(\hat{w}_0, \hat{w}_1)}(x) = \hat{w}_1 x + \hat{w}_0$ that is optimal in the sense of incurring a minimum training error,

$$f(\hat{w}_0, \hat{w}_1) = \min_{w_0, w_1 \in \mathbb{R}} f(w_0, w_1).$$

Let us rewrite the optimality condition (2.28) using matrices and vectors. To this end, we first rewrite the hypothesis (2.24) as

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \text{ with } \mathbf{w} = \big(w_0, w_1\big)^T, \mathbf{x} = \big(1, x\big)^T.$$

Let us stack the feature vectors $\mathbf{x}^{(i)} = \big(1, x^{(i)}\big)^T$ and labels $y^{(i)}$ of the data points (2.23) into a feature matrix and a label vector,

$$\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T \in \mathbb{R}^{m \times 2}, \mathbf{y} = \big(y^{(1)}, \ldots, y^{(m)}\big)^T \in \mathbb{R}^m. \tag{2.29}$$

We can then reformulate (2.28) as

$$\mathbf{X}^T\big(\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}}\big) = \mathbf{0}. \tag{2.30}$$

The optimality conditions (2.30) and (2.28) are equivalent in the following sense. The entries of any parameter vector $\widehat{\mathbf{w}} = \big(\hat{w}_0, \hat{w}_1\big)$ that satisfies (2.30) are solutions to (2.28) and vice-versa.

[2] A necessary and sufficient condition for $\widehat{\mathbf{w}}$ to minimize a convex and differentiable function $f(\mathbf{w})$ is $\nabla f(\widehat{\mathbf{w}}) = \mathbf{0}$ [14, Sec. 4.2.3].

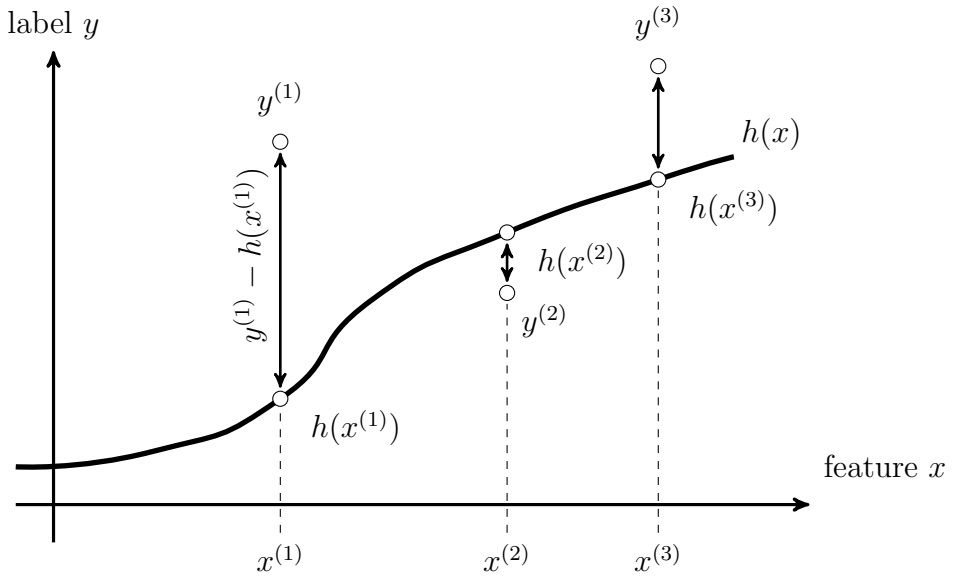


Figure 2.17: We can evaluate the quality of a particular predictor $h \in \mathcal{H}$ by measuring the prediction error $y - h(x)$ obtained for a labeled data point (x, y).

2.5 Exercises

Exercise 2.1. Perfect Prediction. Consider data points that are characterized by a single numeric feature $x \in \mathbb{R}$ and a numeric label $y \in \mathbb{R}$. We use a ML method to learn a hypothesis map $h : \mathbb{R} \to \mathbb{R}$ based on a training set consisting of three data points

$$(x^{(1)} = 1, y^{(1)} = 3), (x^{(2)} = 4, y^{(2)} = -1), (x^{(3)} = 1, y^{(3)} = 5).$$

Is there any chance for the ML method to learn a hypothesis map that perfectly fits the data points such that $h\big(x^{(i)}\big) = y^{(i)}$ for $i = 1, \ldots, 3$. Hint: Try to visualize the data points in a scatterplot and various hypothesis maps (see Figure 1.3).

Exercise 2.2. Temperature Data. Consider a dataset of daily air temperatures $x^{(1)}, \ldots, x^{(m)}$ measured at the FMI observation station "Utsjoki Nuorgam" during 01.12.2019 and 29.02.2020. Thus, $x^{(1)}$ is the daily temperature measured on 01.12.2019, $x^{(2)}$ is the daily temperature on 02.12.2019, and $x^{(m)}$ is the daily temperature measured on 29.02.2020. You can download this dataset from the link `https://en.ilmatieteenlaitos.fi/download-observations`. ML methods often determine few parameters to characterize large collections of data points. Compute, for the above temperature measurement dataset, the following quantities:

- the minimum $A := \min_{i=1,\ldots,m} x^{(i)}$
- the maximum $B := \max_{i=1,\ldots,m} x^{(i)}$
- the average $C := (1/m) \sum_{i=1,\ldots,m} x^{(i)}$
- the standard deviation $D := \sqrt{(1/m) \sum_{i=1,\ldots,m} \big(x^{(i)} - C\big)^2}$

Exercise 2.3. Deep Learning on Raspberry PI. Consider the tiny desktop computer "RaspberryPI" equipped with a total of 8 Gigabytes memory [31]. We want implement a ML algorithm that learns a hypothesis map that is represented by a deep ANN involving $n = 10^6$ numeric parameters. Each parameter is quantized using 8 bits (= 1 Byte). How many different hypotheses can we store at most on a RaspberryPI computer? (You can assume that 1Gigabyte = 10^9Bytes.)

Exercise 2.4. Ensembles. For some applications it can be a good idea to not learn a single hypothesis but to learn a whole ensemble of hypothesis maps $h^{(1)}, \ldots, h^{(B)}$. These hypotheses might even belong to different hypothesis spaces, $h^{(1)} \in \mathcal{H}^{(1)}, \ldots, h^{(B)} \in \mathcal{H}^{(B)}$. These hypothesis spaces can be arbitrary except that they are defined for the same feature space and label space. Given such an ensemble we can construct a new ("meta") hypothesis $\tilde{h}$ by combining (or aggregating) the individual predictions obtained from each hypothesis,

$$\tilde{h}(\mathbf{x}) := a\big(h^{(1)}(\mathbf{x}), \ldots, h^{(B)}(\mathbf{x})\big). \tag{2.31}$$

Here, $a(\cdot)$ denotes some given (fixed) combination or aggregation function. One example for such an aggregation function is the average $a\big(h^{(1)}(\mathbf{x}), \ldots, h^{(B)}(\mathbf{x})\big) := (1/B) \sum_{b=1}^{B} h^{(b)}(\mathbf{x})$. We obtain a new "meta" hypothesis space $\widetilde{\mathcal{H}}$, that consists of all hypotheses of the form (2.31) with $h^{(1)} \in \mathcal{H}^{(1)}, \ldots, h^{(B)} \in \mathcal{H}^{(B)}$. Which conditions on the aggregation function $a(\cdot)$ and the individual hypothesis spaces $\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(B)}$ ensure that $\widetilde{\mathcal{H}}$ contains each individual hypothesis space, i.e., $\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(B)} \subseteq \widetilde{\mathcal{H}}$.

Exercise 2.5. How Many Features? Consider the ML problem underlying a music information retrieval smartphone app [154]. Such an app aims at identifying a song title based on a short audio recording of a song interpretation. Here, the feature vector $\mathbf{x}$ represents the sampled audio signal and the label y is a particular song title out of a huge music database. What is the length n of the feature vector $\mathbf{x} \in \mathbb{R}^n$ whose entries are the

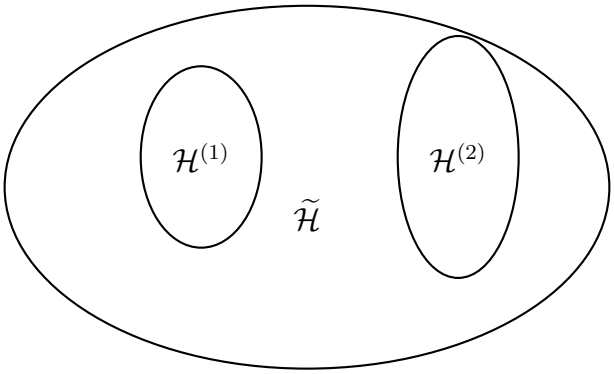


Figure 2.18: The resulting hypothesis space $\widetilde{\mathcal{H}}$ along with some of the constituent hypothesis spaces $\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(B)}$.

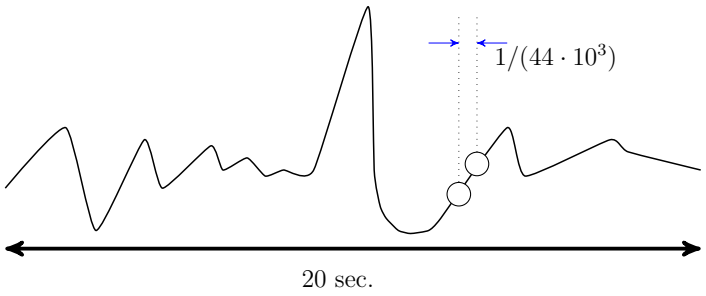


Figure 2.19: A natural choice for the features of an audio recording are the signal amplitudes measured at regular time instants.

signal amplitudes of a 20-second long recording which are sampled at a rate of 44 kHz (see Figure 2.19)?

Exercise 2.6. Multilabel Prediction. Consider data points that are characterized by a feature vector $\mathbf{x} \in \mathbb{R}^{10}$ and a vector-valued label $\mathbf{y} \in \mathbb{R}^{30}$. Such vector-valued labels arise in multi-label classification problems. We want to predict the label vector using a linear predictor map

$$\mathbf{h}(\mathbf{x}) = \mathbf{W}\mathbf{x} \text{ with some matrix } \mathbf{W} \in \mathbb{R}^{30\times 10}. \tag{2.32}$$

How many different linear predictors (2.32) are there ? 10, 30, 40, or infinite?

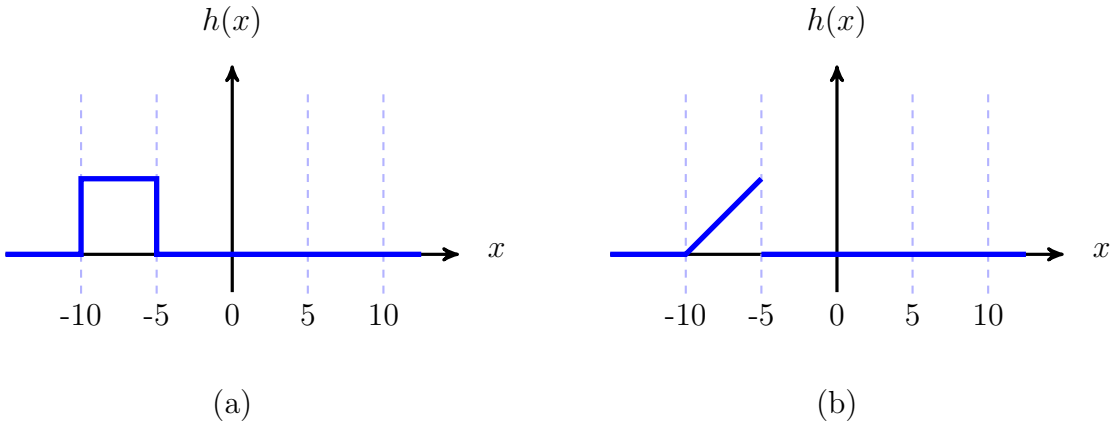
Exercise 2.7. Average Squared Error Loss as Quadratic Form Consider the hypothesis space constituted by all linear maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with some weight vector $\mathbf{w} \in \mathbb{R}^n$. We try to find the best linear map by minimizing the average squared error loss (the empirical risk) incurred on labeled data points (training set) $(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})$. Is it possible to represent the resulting empirical risk as a convex quadratic function $f(\mathbf{w}) = \mathbf{w}^T\mathbf{C}\mathbf{w} + \mathbf{b}\mathbf{w} + c$? If this is possible, how are the matrix $\mathbf{C}$, vector $\mathbf{b}$ and constant c related to the features and labels of data points in the training set?

Exercise 2.8. Find Labeled Data for Given Empirical Risk. Consider linear hypothesis space consisting of linear maps $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that are parametrized by a weight vector $\mathbf{w}$. We learn an optimal weight vector by minimizing the average squared error loss $f(\mathbf{w}) = \widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)$ incurred by $h^{(\mathbf{w})}(\mathbf{x})$ on the training set $\mathcal{D} = \big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)$. Is it possible to reconstruct the dataset $\mathcal{D}$ just from knowing the function $f(\mathbf{w})$?. Is the resulting labeled training data unique or are there different training sets that could have resulted in the same empirical risk function? Hint: Write down the training error $f(\mathbf{w})$ in the form $f(\mathbf{w}) = \mathbf{w}^T\mathbf{Q}\mathbf{w} + c + \mathbf{b}^T\mathbf{w}$ with some matrix $\mathbf{Q}$, vector $\mathbf{b}$ and scalar c that might depend on the features and labels of the training data points.

Exercise 2.9. Dummy Feature Instead of Intercept. Show that any hypothesis map of the form $h(x) = w_1x + w_0$ can be obtained from the concatenation of a feature map $\phi : x \mapsto \mathbf{z}$ with the linear map $\tilde{h}(\mathbf{z}) := \widetilde{\mathbf{w}}^T\mathbf{z}$ using parameter vector $\widetilde{\mathbf{w}} = \big(w_1, w_0\big)^T \in \mathbb{R}^2$.

Exercise 2.10. Approximate Non-Linear Maps Using Indicator Functions for Feature Maps. Consider an ML application generating data points characterized by a scalar feature $x \in \mathbb{R}$ and numeric label $y \in \mathbb{R}$. We construct a non-linear map by first transforming the feature x to a new feature vector $\mathbf{z} = (\phi_1(x), \phi_2(x), \phi_3(x), \phi_4(x))^T \in \mathbb{R}^4$. The

components $\phi_1(x), \ldots, \phi_4(x)$ are indicator functions of intervals $[-10,-5), [-5,0), [0,5), [5,10]$. In particular, $\phi_1(x) = 1$ for $x \in [-10,-5)$ and $\phi_1(x) = 0$ otherwise. We obtain a hypothesis space $\mathcal{H}^{(1)}$ by collecting all maps from feature x to predicted label $\hat{y}$ that can written as a a weighted linear combination $\mathbf{w}^T\mathbf{z}$ (with some parameter vector $\mathbf{w}$) of the transformed features. Which of the following hypothesis maps belong to $\mathcal{H}^{(1)}$?



Exercise 2.11. Python Hypothesis space. Consider the source codes below for five different Python functions that read in the numeric feature x, perform some computations that result in a prediction $\hat{y}$. How large is the hypothesis space that is constituted by all maps that can be represented by one of those Python functions.

```
def func1(x):
hat_y = 5*x+3
return hat_y

```

```
def func2(x):
tmp = 3*x+3
hat_y = tmp+2*x
return hat_y

```

```
def func3(x):
tmp = 3*x+3
hat_y = tmp-2*x
return hat_y

```

```
def func4(x):
tmp = 3*x+3
hat_y = tmp-2*x+4
return hat_y

```

```
def func5(x):
tmp = 3*x+3
hat_y = 4*tmp-2*x
return hat_y

```

Exercise 2.12. A Lot of Features. One important application domain for ML methods is healthcare. Here, data points represent human patients that are characterized by health-care records. These records might contain physiological parameters, CT scans along with various diagnoses provided by healthcare professionals. Is it a good idea to use every data field of a healthcare record as features of the data point ?

Exercise 2.13. Over-Parametrization. Consider data points characterized by feature vectors $\mathbf{x} \in \mathbb{R}^2$ and a numeric label $y \in \mathbb{R}$. We want to learn the best predictor out of the hypothesis space

$$\mathcal{H} = \big\{h(\mathbf{x}) = \mathbf{x}^T \mathbf{A}\mathbf{w} : \mathbf{w} \in \mathcal{S}\big\}.$$

Here, we used the matrix $\mathbf{A} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ and the set

$$\mathcal{S} = \big\{(1,1)^T, (2,2)^T, (-1,3)^T, (0,4)^T\big\} \subseteq \mathbb{R}^2.$$

What is the cardinality of the hypothesis space $\mathcal{H}$, i.e., how many different predictor maps does $\mathcal{H}$ contain?

Exercise 2.14. Squared Error Loss. Consider a hypothesis space $\mathcal{H}$ constituted by three predictors $h^{(1)}(\cdot), h^{(2)}(\cdot), h^{(3)}(\cdot)$. Each predictor $h^{(j)}(x)$ is a real-valued function of a real-valued argument x. Moreover, for each $j \in \{1,2,3\}$,

$$h^{(j)}(x) = \begin{cases} 0 & \text{if } x^2 \leq j \\ j & \text{otherwise.} \end{cases} \tag{2.33}$$

Can you tell which of these hypothesis maps is optimal in the sense of having smallest average squared error loss on the dataset

$$\big\{\, (1/10, 3)\,, (0,0)\,, (1,-1)\,\big\}.$$

Exercise 2.15. Classification Loss. The Figure 2.16 depicts different loss functions for a fixed data point with label $y = 1$ and varying hypothesis $h \in \mathcal{H}$. How would Figure 2.16 change if we evaluate the same loss functions for another data point $\mathbf{z} = (x, y)$ with label $y = -1$?

Exercise 2.16. Intercept Term. Linear regression methods model the relation between the label y and feature x of a data point as $y = h(x) + e$ with some small additive term e. The predictor map $h(x)$ is assumed to be linear $h(x) = w_1 x + w_0$. The parameter w_0 is sometimes referred to as the intercept (or bias) term. Assume we know for a given linear predictor map its values $h(x)$ for $x = 1$ and $x = 3$. Can you determine the weights w_1 and

w_0 based on $h(1)$ and $h(3)$?

Exercise 2.17. Picture Classification. Consider a huge collection of outdoor pictures you have taken during your last adventure trip. You want to organize these pictures as three categories (or classes) *dog*, *bird* and *fish*. How could you formalize this task as a ML problem?

Exercise 2.18. Maximal Hypothesis space. Consider data points characterized by a single real-valued feature x and a single real-valued label y. How large is the largest possible hypothesis space of predictor maps $h(x)$ that read in the feature value of a data point and deliver a real-valued prediction $\hat{y} = h(x)$?

Exercise 2.19. A Large but Finite Hypothesis space. Consider data points whose features are 10×10 pixel black-and-white (bw) images. Besides the pixels, each data point is also characterized by a binary label $y \in \{0, 1\}$. Consider the hypothesis space which is constituted by all maps that take a 10×10 pixel bw image as input and deliver a prediction for the label. How large is this hypothesis space?

Exercise 2.20. Size of Linear Hypothesis space. Consider a training set of m data points with feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and numeric labels $y^{(1)}, \ldots, y^{(m)}$. The feature vectors and labels of the data points in the training set are arbitrary except that we assume the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)$ is full rank. What condition on m and n guarantees that we can find a linear predictor $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that perfectly fits the training set, i.e., $y^{(1)} = h\big(\mathbf{x}^{(1)}\big), \ldots, y^{(m)} = h\big(\mathbf{x}^{(m)}\big)$.

Exercise 2.21. Classification with Imbalanced Data. Consider a dataset $\mathcal{D}$ of m data points with feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and discrete-valued labels $y^{(i)} \in \{1, 2, \ldots, 10\}$. The data is highly imbalanced, more than 90 percent of data points have a label $y = 1$. We learn a hypothesis out of the hypothesis space $\mathcal{H}'$ that is constituted by the ten maps $h^{(c)}(\mathbf{x}) = c$ for $c = 1, 2, \ldots, 10$. Is there a hypothesis $h \in \mathcal{H}'$ whose average 0/1 loss on $\mathcal{D}$ does not exceed 0.3 ?

Exercise 2.22. Accuracy and Average Logistic loss. Consider a dataset $\mathcal{D}$ that consists of m data points , indexed by $i = 1, \ldots, m$. Each data point is characterized by a feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^2$ and by a binary label $y^{(i)} \in \{-1, 1\}$. We use a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ to classify data points according to $\hat{y} = 1$ if $h^{(\mathbf{w})}(\mathbf{x}) \geq 0$ and $\hat{y} = -1$

otherwise. Two popular quality measures of a hypothesis are the accuracy and the average logistic loss $(1/m)\sum_i L\left(\left(\mathbf{x}^{(i)}, y^{(i)}\right), h^{(\mathbf{w})}\right)$ with the logistic loss (2.12).

The accuracy of a hypothesis $h^{(\mathbf{w})}(\mathbf{x})$ is defined as $1-(1/m)\sum_i L\left(\left(\mathbf{x}^{(i)}, y^{(i)}\right), h^{(\mathbf{w})}\right)$ with the 0/1 loss (2.9). Loosely speaking, the accuracy of a hypothesis is "one minus the average 0/1 loss ". Can you construct a specific dataset with arbitrary but finite size m such that there are two different linear hypotheses $h^{(\mathbf{w})}$ and $h^{(\mathbf{w}')}$ with accuracy and average logistic loss of $h^{(\mathbf{w})}$, respectively, being strictly larger than accuracy and average logistic loss of $h^{(\mathbf{w}')}$.

Exercise 2.23. Achievable Accuracy for a Multi-Class Classification Problem. Consider a dataset consisting of data points, each characterized by some features $\mathbf{x}$ and label $y \in \{1,2,3,4,5\}$. The number of data points with label value $y=1$ is 150, those with $y=2$ is 250, with $y=3$ is 200, with $y=4$ is 300 and the number of data points with $y=5$ is 100. Is there at least one classifier that can achieve an accuracy of at least 10 %?

Exercise 2.24. Make it Easier? Consider a multi-class classification problem with data points belonging to one out of 10 different classes represented by the label space $\mathcal{Y} = \{1,2,3,\ldots,10\}$. Assume that we have found a hypothesis map (classifier) that achieves an accuracy of 70 %. Now consider a modified multi-class classification problem that is obtained by combining 4 classes of $\mathcal{Y}$ into one single new class. Can we be sure to find a classifier for this new problem that achieves an accuracy of at least 70 % ?

Chapter 3

The Landscape of ML

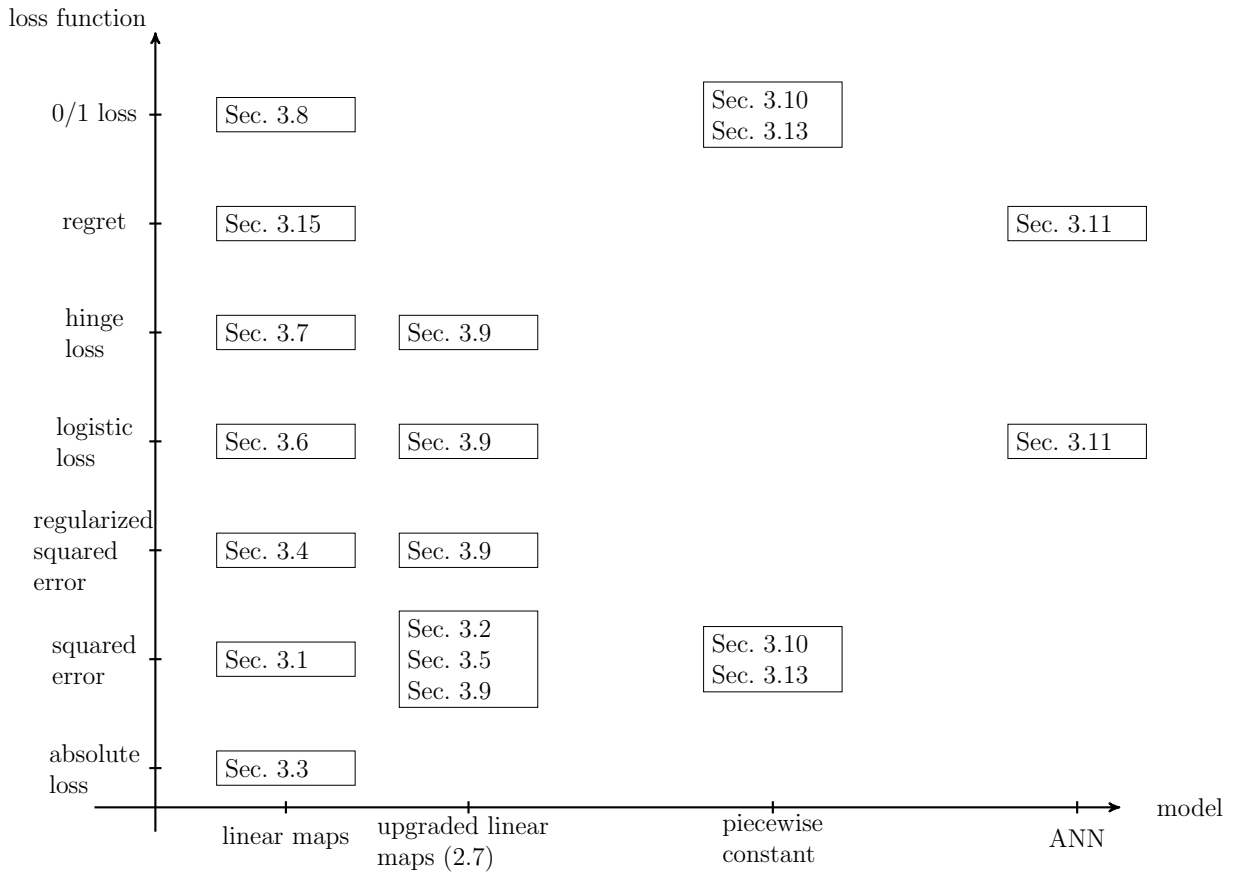


Figure 3.1: ML methods fit a model to data by minimizing a loss function. Different ML methods use different design choices for data, model and loss.

As discussed in Chapter 2, ML methods combine three main components:

- a set of data points that are characterized by features and labels
- a model or hypothesis space $\mathcal{H}$ that consists of different hypotheses $h \in \mathcal{H}$.
- a loss function to measure the quality of a particular hypothesis h.

Each of these three components involves design choices for the representation of data, their features and labels, the model and loss function. This chapter details the high-level design choices used by some of the most popular ML methods. Figure 3.1 depicts these ML methods in a two-dimensional plane whose horizontal axes represents different hypothesis spaces and the vertical axis represents different loss functions.

To obtain a practical ML method we also need to combine the above components. The basic principle of any ML method is to search the model for a hypothesis that incurs minimum loss on any data point. Chapter 4 will then discuss a principled way to turn this informal statement into actual ML algorithms that could be implemented on a computer.

3.1 Linear Regression

Consider data points characterized by feature vectors $\mathbf{x} \in \mathbb{R}^n$ and numeric label $y \in \mathbb{R}$. Linear regression methods learn a hypothesis out of the linear hypothesis space

$$\mathcal{H}^{(n)} := \{h^{(\mathbf{w})} : \mathbb{R}^n \to \mathbb{R} : h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \text{ with some parameter vector } \mathbf{w} \in \mathbb{R}^n\}. \tag{3.1}$$

Figure 1.3 depicts the graphs of some maps from $\mathcal{H}^{(2)}$ for data points with feature vectors of the form $\mathbf{x} = (1, x)^T$. The quality of a particular predictor $h^{(\mathbf{w})}$ is measured by the squared error loss (2.8). Using labeled data $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$, linear regression learns a linear predictor $\widehat{h}$ which minimizes the average squared error loss (2.8),

$$\widehat{h} = \operatorname*{argmin}_{h \in \mathcal{H}^{(n)}} \widehat{L}(h|\mathcal{D}) \overset{(2.16)}{=} \operatorname*{argmin}_{h \in \mathcal{H}^{(n)}} (1/m) \sum_{i=1}^{m} (y^{(i)} - h(\mathbf{x}^{(i)}))^2. \tag{3.2}$$

Some authors refer to the average squared error loss in (3.2) also as the mean squared error of the hypothesis h.

Since the hypothesis space $\mathcal{H}^{(n)}$ is parametrized by the parameter vector $\mathbf{w}$ (see (3.1)),

we can rewrite (3.2) as an optimization of the parameter vector $\mathbf{w}$,

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} (1/m) \sum_{i=1}^{m} (y^{(i)} - h^{(\mathbf{w})}(\mathbf{x}^{(i)}))^2$$
$$\overset{h^{(\mathbf{w})}(\mathbf{x})=\mathbf{w}^T\mathbf{x}}{=} \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} (1/m) \sum_{i=1}^{m} (y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)})^2. \tag{3.3}$$

The optimization problems (3.2) and (3.3) are equivalent in the following sense: Any optimal parameter vector $\widehat{\mathbf{w}}$ which solves (3.3), can be used to construct an optimal predictor $\widehat{h}$, which solves (3.2), via $\widehat{h}(\mathbf{x}) = h^{(\widehat{\mathbf{w}})}(\mathbf{x}) = \big(\widehat{\mathbf{w}}\big)^T\mathbf{x}$.

3.2 Polynomial Regression

Consider an ML problem involving data points which are characterized by a single numeric feature $x \in \mathbb{R}$ (the feature space is $\mathcal{X} = \mathbb{R}$) and a numeric label $y \in \mathbb{R}$ (the label space is $\mathcal{Y} = \mathbb{R}$). We observe a bunch of labeled data points which are depicted in Figure 3.2.

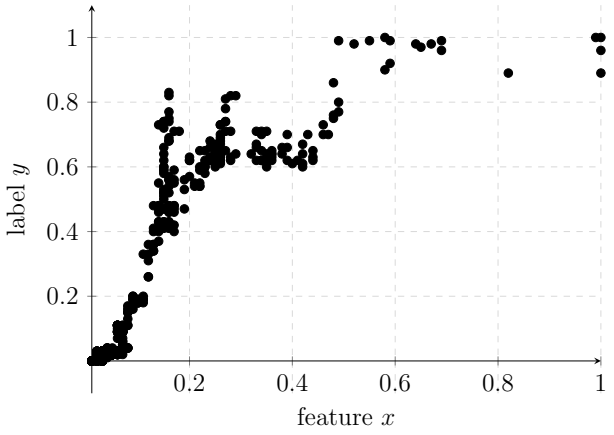


Figure 3.2: A scatterplot that depicts a set of data points $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$. The ith data point is depicted by a dot whose coordinates are the feature $x^{(i)}$ and label $y^{(i)}$ of that data point.

Figure 3.2 suggests that the relation $x \mapsto y$ between feature x and label y is highly non-linear. For such non-linear relations between features and labels it is useful to consider a

hypothesis space which is constituted by polynomial maps

$$\mathcal{H}^{(n)}_{\rm poly} = \{h^{(\mathbf{w})} : \mathbb{R} \to \mathbb{R} : h^{(\mathbf{w})}(x) = \sum_{j=1}^{n} w_j x^{j-1}, \\ \text{with some } \mathbf{w} = (w_1, \ldots, w_n)^T \in \mathbb{R}^n\}. \tag{3.4}$$

We can approximate any non-linear relation $y = h(x)$ with any desired level of accuracy using a polynomial $\sum_{j=1}^{n} w_j x^{j-1}$ of sufficiently large degree $n-1$.[1]

As in linear regression (see Section 3.1), we measure the quality of a hypothesis by the squared error loss (2.8) incurred on labeled data points $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, each having a scalar feature $x^{(i)}$ and label $y^{(i)}$. Polynomial regression learns a hypothesis h by minimizing the average squared error loss (see (2.8)):

$$\min_{h^{(\mathbf{w})} \in \mathcal{H}^{(n)}_{\rm poly}} (1/m) \sum_{i=1}^{m} (y^{(i)} - h^{(\mathbf{w})}(x^{(i)}))^2. \tag{3.5}$$

We can interpret polynomial regression as a combination of a feature map (transformation) (see Section 2.1.1) and linear regression (see Section 3.1). Indeed, any polynomial predictor $h^{(\mathbf{w})} \in \mathcal{H}^{(n)}_{\rm poly}$ is obtained as a concatenation of the feature map

$$\phi(x) \mapsto \mathbf{x} := (1, x, \ldots, x^{n-1})^T \in \mathbb{R}^n \tag{3.6}$$

with some linear map $\tilde{h}^{(\mathbf{w})} : \mathbb{R}^n \to \mathbb{R} : \mathbf{x} \mapsto \mathbf{w}^T \mathbf{x}$, i.e.,

$$h^{(\mathbf{w})}(x) = \tilde{h}^{(\mathbf{w})}(\underbrace{\phi(x)}_{\overset{(3.6)}{=}\mathbf{x}}). \tag{3.7}$$

Thus, we can implement polynomial regression by first applying the feature map ϕ (see (3.6)) to the scalar features $x^{(i)}$, resulting in the transformed feature vectors

$$\mathbf{x}^{(i)} = \phi\big(x^{(i)}\big) = \big(1, x^{(i)}, \ldots, \big(x^{(i)}\big)^{n-1}\big)^T \in \mathbb{R}^n, \tag{3.8}$$

and then applying linear regression (see Section 3.1) to these transformed feature vectors $\mathbf{x}^{(i)}$.

[1] The precise formulation of this statement is known as the "Stone-Weierstrass Theorem" [119, Thm. 7.26].

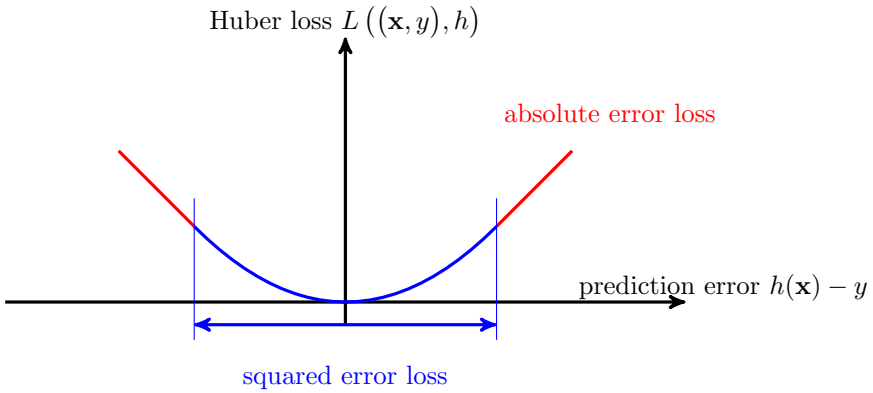


Figure 3.3: The Huber loss (3.9) resembles the squared error loss (2.8) for small prediction error and the absolute difference loss for larger prediction errors.

By inserting (3.7) into (3.5), we obtain a linear regression problem (3.3) with feature vectors (3.8). Note that while the hypothesis $h^{(\mathbf{w})} \in \mathcal{H}^{(n)}_{\rm poly}$ is a non-linear function $h^{(\mathbf{w})}(x)$ of the original feature x, it is a linear function $\tilde{h}^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ (see (3.7)), of the transformed features $\mathbf{x}$ (3.8).

3.3 Least Absolute Deviation Regression

Learning a linear predictor by minimizing the average squared error loss incurred on training data is not robust against the presence of outliers. This sensitivity to outliers is rooted in the properties of the squared error loss $(y-h(\mathbf{x}))^2$. Minimizing the average squared error forces the resulting predictor $\hat{y}$ to not be too far away from any data point. However, it might be useful to tolerate a large prediction error $y-h(\mathbf{x})$ for an unusual or exceptional data point that can be considered an outlier.

Replacing the squared loss with a different loss function can make the learning robust against outliers. One important example for such a "robustifying" loss function is the Huber loss [63]

$$L\left((\mathbf{x},y),h\right) = \begin{cases} (1/2)(y-h(\mathbf{x}))^2 & \text{for } |y-h(\mathbf{x})| \leq \varepsilon \\ \varepsilon(|y-h(\mathbf{x})| - \varepsilon/2) & \text{else.} \end{cases} \tag{3.9}$$

Figure 3.3 depicts the Huber loss as a function of the prediction error $y-h(\mathbf{x})$.

The Huber loss definition (3.9) contains a tuning parameter ϵ. The value of this tuning parameter defines when a data point is considered as an outlier. Figure 3.4 illustrates the

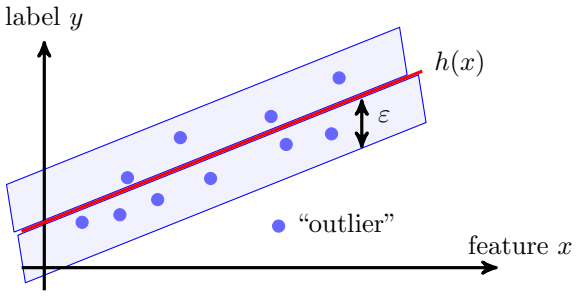


Figure 3.4: The Huber loss measures prediction errors via squared error loss for regular data points inside the band of width ε around the hypothesis map $h(\mathbf{x})$ and via the absolute difference loss for an outlier outside the band.

role of this parameter as the width of a band around a hypothesis map. The prediction error of this hypothesis map for data points within this band are measured used squared error loss (2.8). For data points outside this band (these are considered outliers), we use instead the absolute value of the prediction error as the loss value.

The Huber loss is robust to outliers since the corresponding (large) prediction errors $y - \hat{y}$ are not squared. Outliers have a smaller effect on the average Huber loss (over the entire dataset) compared to the average squared error loss. The improved robustness against outliers of the Huber loss comes at the expense of increased computational complexity. The squared error loss can be minimized using efficient gradient based methods (see Chapter 5). In contrast, for $\varepsilon = 0$, the Huber loss is non-differentiable and requires more advanced optimization methods.

The Huber loss (3.9) contains two important special cases. The first special case occurs when ε is chosen to be very large, such that the condition $|y - \hat{y}| \leq \varepsilon$ is satisfied for most data points. In this case, the Huber loss resembles the squared error loss (2.8) (up to a scaling factor $1/2$). The second special case is obtained for $\varepsilon \to 0$ when the scaled Huber loss $(1/\varepsilon)L\big((\mathbf{x}, y), h\big)$ converges to the absolute difference loss $|y - \hat{y}|$.

3.4 The Lasso

We will see in Chapter 6 that linear regression (see Section 3.1) typically requires a training set larger than the number of features used to characterized a data point. However, many important application domains generate data points with a number n of features much higher than the number m of available labeled data points in the training set.

In the high-dimensional regime, where $m \ll n$, basic linear regression methods will not be able to learn useful weights $\mathbf{w}$ for a linear hypothesis. Section 6.4 shows that for $m \ll n$, linear regression will typically learn a hypothesis that perfectly predicts labels of data points in the training set but delivers poor predictions for data points outside the training set. This phenomenon is referred to as overfitting and poses a main challenge for ML applications in the high-dimensional regime.

Chapter 7 discusses basic regularization techniques that allow to prevent ML methods from overfitting. We can regularize linear regression by augmenting the squared error loss (2.8) of a hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with an additional penalty term. This penalty term depends solely on the weights $\mathbf{w}$ and serves as an estimate for the increase of the average loss on data points outside the training set. Different ML methods are obtained from different choices for this penalty term. The least absolute shrinkage and selection operator (Lasso) is obtained from linear regression by replacing the squared error loss with the regularized loss

$$L\left((\mathbf{x}, y), h^{(\mathbf{w})}\right) = \left(y - \mathbf{w}^T\mathbf{x}\right)^2 + \lambda\|\mathbf{w}\|_1. \tag{3.10}$$

Here, the penalty term is given by the scaled norm $\lambda\|\mathbf{w}\|_1$ with the regularization parameter λ. The value of λ has significant impact on the properties of the hypothesis learnt by minimizing (3.10). Roughly speaking, increasing λ results in a weight vector $\widehat{\mathbf{w}}$ with increasing number of zero coefficients. Thus, the learnt hypothesis $h^{(\widehat{\mathbf{w}})}$ ignores many features, which is a form of feature selection.

One principled approach to finding a suitable value range for λ can be based on the i.i.d. assumption (Section 2.1.4). We assume that the features and label of a data point can be modelled as

$$y = \overline{\mathbf{w}}^T\mathbf{x} + \varepsilon. \tag{3.11}$$

Here, $\overline{\mathbf{w}}$ denotes some true underlying parameter vector and ε is a realization of an a RV that is independent of the features $\mathbf{x}$. It is then possible to determine optimal values for λ in (3.10) such that the learnt weights $\widehat{\mathbf{w}}$ are close to the (unknown) true weights $\overline{\mathbf{w}}$ [151, 17]. Note that the optimal values for λ depend crucially the distribution of the noise ε and the number of non-zeros entries in $\overline{\mathbf{w}}$ in (3.11).

Instead of relying on a probabilistic model (3.11), which might be wrong, we can also tune λ by a trial-and-error approach. Here, we try out different candidate values for λ and learn (the weights of) a linear hypothesis for each value of λ. Among those, we pick the hypothesis resulting in the minimum average loss on a validation set. This approach is an

instance of the model selection techniques in Section 6.3.

3.5 Gaussian Basis Regression

Section 3.2 showed how to extend linear regression by first transforming the feature x using a vector-valued feature map $\mathbf{\Phi} : \mathbb{R} \to \mathbb{R}^n$. The output of this feature map are the transformed features $\mathbf{\Phi}(x)$ which are fed, in turn, to a linear map $h\left(\mathbf{\Phi}(x)\right) = \mathbf{w}^T\phi(x)$. Polynomial regression in Section 3.2 has been obtained for the specific feature map (3.6) whose entries are the powers x^l of the scalar original feature x. However, it is possible to use other functions, different from polynomials, to construct the feature map ϕ. We can extend linear regression using an arbitrary feature map

$$\mathbf{\Phi}(x) = (\phi_1(x), \ldots, \phi_n(x))^T \tag{3.12}$$

with the scalar maps $\phi_j : \mathbb{R} \to \mathbb{R}$ which are referred to as "basis functions". The choice of basis functions depends heavily on the particular application and the underlying relation between features and labels of the observed data points. The basis functions underlying polynomial regression are $\phi_j(x) = x^j$.

Another popular choice for the basis functions are "Gaussians"

$$\phi_{\sigma,\mu}(x) = \exp(-(1/(2\sigma^2))(x-\mu)^2). \tag{3.13}$$

The family (3.13) of maps is parameterized by the variance σ^2 and the mean μ. Gaussian basis linear regression combines the feature map

$$\mathbf{\Phi}(x) = \big(\phi_{\sigma_1,\mu_1}(x), \ldots, \phi_{\sigma_n,\mu_n}(x)\big)^T \tag{3.14}$$

with linear regression (see Figure 3.5). The resulting hypothesis space is

$$\begin{aligned}\mathcal{H}^{(n)}_{\text{Gauss}} = \{h^{(\mathbf{w})} : \mathbb{R} \to \mathbb{R} : h^{(\mathbf{w})}(x) &= \sum_{j=1}^{n} w_j \phi_{\sigma_j,\mu_j}(x) \\ &\text{with weights } \mathbf{w} = (w_1, \ldots, w_n)^T \in \mathbb{R}^n\}.\end{aligned} \tag{3.15}$$

Different choices for the variance σ_j^2 and shifts μ_j of the Gaussian function in (3.13) results in different hypothesis spaces $\mathcal{H}_{\text{Gauss}}$. Chapter 6.3 will discuss model selection techniques

that allow to find useful values for these parameters.

The hypotheses of (3.15) are parametrized by a parameter vector $\mathbf{w} \in \mathbb{R}^n$. Each hypothesis in $\mathcal{H}_{\text{Gauss}}$ corresponds to a particular choice for the parameter vector $\mathbf{w}$. Thus, instead of searching over $\mathcal{H}_{\text{Gauss}}$ to find a good hypothesis, we can search over the Euclidean space $\mathbb{R}^n$. Highly developed methods for searching over the space $\mathbb{R}^n$, for a wide range of values for n, are provided by numerical linear algebra [46].

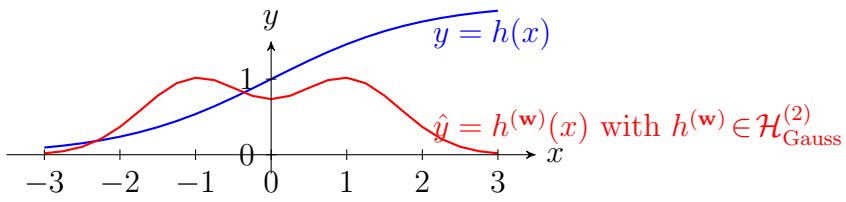


Figure 3.5: The true relation $x \mapsto y$ (blue) between feature x and label y of data points is highly non-linear. Therefore it seems reasonable to predict the label using a non-linear hypothesis map $h^{(\mathbf{w})}(x) \in \mathcal{H}^{(2)}_{\text{Gauss}}$ with some parameter vector $\mathbf{w} \in \mathbb{R}^2$.

3.6 Logistic Regression

Logistic regression is a ML method that allows to classify data points according to two categories. Thus, logistic regression is a binary classification method that can be applied to data points characterized by feature vectors $\mathbf{x} \in \mathbb{R}^n$ (feature space $\mathcal{X} = \mathbb{R}^n$) and binary labels y. These binary labels take on values from a label space that contains two different label values. Each of these two label values represents one of the two categories to which the data points can belong.

It is convenient to use the label space $\mathcal{Y} = \mathbb{R}$ and encode the two label values as $y = 1$ and $y = -1$. However, it is important to note that logistic regression can be used with an arbitrary label space which contains two different elements. Another popular choice for the label space is $\mathcal{Y} = \{0, 1\}$. Logistic regression learns a hypothesis out of the hypothesis space $\mathcal{H}^{(n)}$ (see (3.1)). Note that logistic regression uses the same hypothesis space as linear regression (see Section 3.1).

At first sight, it seems wasteful to use a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, with some parameter vector $\mathbf{w} \in \mathbb{R}^n$, to predict a binary label y. Indeed, while the prediction $h(\mathbf{x})$ can take any real number, the label $y \in \{-1, 1\}$ takes on only one of the two real numbers 1 and -1.

It turns out that even for binary labels it is quite useful to use a hypothesis map h which can take on arbitrary real numbers. We can always obtain a predicted label $\hat{y} \in \{-1,1\}$ by comparing hypothesis value $h(\mathbf{x})$ with a threshold. A data point with features $\mathbf{x}$, is classified as $\hat{y}=1$ if $h(\mathbf{x}) \geq 0$ and $\hat{y}=-1$ for $h(\mathbf{x})<0$. Thus, we use the sign of the hypothesis h to determine the classification result, which is the prediction $\hat{y} \in \mathcal{Y}$ for the true label $y \in \mathcal{Y}$. The absolute value $|h(\mathbf{x})|$ is then used to quantify the reliability of (confidence in) the classification $\hat{y}$.

Consider two data points with feature vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ and a linear classifier map h yielding the function values $h(\mathbf{x}^{(1)})=1/10$ and $h(\mathbf{x}^{(2)})=100$. Whereas the predictions for both data points result in the same label predictions, i.e., $\hat{y}^{(1)}=\hat{y}^{(2)}=1$, the classification of the data point with feature vector $\mathbf{x}^{(2)}$ seems to be much more reliable.

Logistic regression uses the logistic loss (2.12) to assess the quality of a particular hypothesis $h^{(\mathbf{w})} \in \mathcal{H}^{(n)}$. In particular, given some labeled training set $\mathcal{D}=\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{m}$, logistic regression tries to minimize the empirical risk (average logistic loss)

$$\begin{aligned}\widehat{L}(\mathbf{w}|\mathcal{D}) &= (1/m)\sum_{i=1}^{m} \log(1+\exp(-y^{(i)}h^{(\mathbf{w})}(\mathbf{x}^{(i)}))) \\ &\overset{h^{(\mathbf{w})}(\mathbf{x})=\mathbf{w}^T\mathbf{x}}{=} (1/m)\sum_{i=1}^{m} \log(1+\exp(-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)})). \end{aligned} \tag{3.16}$$

Once we have found the optimal parameter vector $\widehat{\mathbf{w}}$, which minimizes (3.16), we can classify any data point solely based on its features $\mathbf{x}$. Indeed, we just need to evaluate the hypothesis $h^{(\widehat{\mathbf{w}})}$ for the features $\mathbf{x}$ to obtain the predicted label

$$\hat{y} = \begin{cases} 1 & \text{if } h^{(\widehat{\mathbf{w}})}(\mathbf{x}) \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{3.17}$$

Since $h^{(\widehat{\mathbf{w}})}(\mathbf{x}) = \big(\widehat{\mathbf{w}}\big)^T\mathbf{x}$ (see (3.1)), the classifier (3.17) amounts to testing whether $\big(\widehat{\mathbf{w}}\big)^T\mathbf{x} \geq 0$ or not.

The classifier (3.17) partitions the feature space $\mathcal{X}=\mathbb{R}^n$ into two half-spaces $\mathcal{R}_1=\big\{\mathbf{x}: \big(\widehat{\mathbf{w}}\big)^T\mathbf{x}\geq 0\big\}$ and $\mathcal{R}_{-1}=\big\{\mathbf{x}: \big(\widehat{\mathbf{w}}\big)^T\mathbf{x}<0\big\}$ which are separated by the hyperplane $\big(\widehat{\mathbf{w}}\big)^T\mathbf{x}=0$ (see Figure 2.9). Any data point with features $\mathbf{x} \in \mathcal{R}_1$ ($\mathbf{x} \in \mathcal{R}_{-1}$) is classified as $\hat{y}=1$ ($\hat{y}=-1$).

Logistic regression can be interpreted as a statistical estimation method for a particular probabilistic model for the data points. This probabilistic model interprets the label $y \in$

$\{-1, 1\}$ of a data point as a RV with the probability distribution

$$\begin{aligned} p(y=1;\mathbf{w}) &= 1/(1+\exp(-\mathbf{w}^T\mathbf{x})) \\ &\overset{h^{(\mathbf{w})}(\mathbf{x})=\mathbf{w}^T\mathbf{x}}{=} 1/(1+\exp(-h^{(\mathbf{w})}(\mathbf{x}))). \end{aligned} \tag{3.18}$$

As the notation indicates, the probability (3.18) is parametrized by the parameter vector $\mathbf{w}$ of the linear hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Given the probabilistic model (3.18), we can interpret the classification (3.17) as choosing $\hat{y}$ to maximize the probability $p(y = \hat{y}; \mathbf{w})$.

Since $p(y = 1) + p(y = -1) = 1$,

$$\begin{aligned} p(y=-1) &= 1 - p(y=1) \\ &\overset{(3.18)}{=} 1 - 1/(1+\exp(-\mathbf{w}^T\mathbf{x})) \\ &= 1/(1+\exp(\mathbf{w}^T\mathbf{x})). \end{aligned} \tag{3.19}$$

In practice we do not know the parameter vector in (3.18). Rather, we have to estimate the parameter vector $\mathbf{w}$ in (3.18) from observed data points. A principled approach to estimate the parameter vector is to maximize the probability (or likelihood) of actually obtaining the dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$ as realizations of i.i.d. data points whose labels are distributed according to (3.18). This yields the maximum likelihood estimator

$$\begin{aligned} \widehat{\mathbf{w}} &= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^n} p(\{y^{(i)}\}_{i=1}^m) \\ &\overset{y^{(i)}\text{i.i.d.}}{=} \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^n} \prod_{i=1}^m p(y^{(i)}) \\ &\overset{(3.18),(3.19)}{=} \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^n} \prod_{i=1}^m 1/(1+\exp(-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)})). \end{aligned} \tag{3.20}$$

Note that the last expression (3.20) is only valid if we encode the binary labels using the values 1 and -1. Using different label values results in a different expression.

Maximizing a positive function $f(\mathbf{w}) > 0$ is equivalent to maximizing $\log f(w)$,

$$\operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}) = \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^n} \log f(\mathbf{w}).$$

Therefore, (3.20) can be further developed as

$$\widehat{\mathbf{w}} \stackrel{(3.20)}{=} \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^n} \sum_{i=1}^{m} -\log\big(1+\exp(-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)})\big)$$
$$= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} (1/m) \sum_{i=1}^{m} \log\big(1+\exp(-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)})\big). \quad (3.21)$$

Comparing (3.21) with (3.16) reveals that logistic regression is nothing but maximum likelihood estimation of the parameter vector $\mathbf{w}$ in the probabilistic model (3.18).

3.7 Support Vector Machines

SVMs are a family of ML methods for learning a hypothesis to predict a binary label y of a data point based on its features $\mathbf{x}$. Without loss of generality we consider binary labels taking values in the label space $\mathcal{Y} = \{-1, 1\}$. A SVM uses the linear hypothesis space (3.1) which consists of linear maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with some parameter vector $\mathbf{w} \in \mathbb{R}^n$. Thus, the SVM uses the same hypothesis space as linear regression and logistic regression which we have discussed in Section 3.1 and Section 3.6, respectively. What sets the SVM apart from these other methods is the choice of loss function.

Different instances of a SVM are obtained by using different constructions for the features of a data point. Kernel SVMs use the concept of a kernel map to construct (typically high-dimensional) features (see Section 3.9 and [82]). In what follows, we assume the feature construction has been solved and we have access to a feature vector $\mathbf{x} \in \mathbb{R}^n$ for each data point.

Figure 3.6 depicts a dataset $\mathcal{D}$ of labeled data points, each characterized by a feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^2$ (used as coordinates of a marker) and a binary label $y^{(i)} \in \{-1, 1\}$ (indicated by different marker shapes). We can partition dataset $\mathcal{D}$ into two classes

$$\mathcal{C}^{(y=1)} = \{\mathbf{x}^{(i)} : y^{(i)} = 1\}, \text{ and } \mathcal{C}^{(y=-1)} = \{\mathbf{x}^{(i)} : y^{(i)} = -1\}. \quad (3.22)$$

The SVM tries to learn a linear map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that perfectly separates the two classes in the sense of

$$\underbrace{h\big(\mathbf{x}^{(i)}\big)}_{\mathbf{w}^T\mathbf{x}^{(i)}} > 0 \text{ for } \mathbf{x}^{(i)} \in \mathcal{C}^{(y=1)} \text{ and } \underbrace{h\big(\mathbf{x}^{(i)}\big)}_{\mathbf{w}^T\mathbf{x}^{(i)}} < 0 \text{ for } \mathbf{x}^{(i)} \in \mathcal{C}^{(y=-1)}. \quad (3.23)$$

We refer to a dataset, whose data points have binary labels. as linear separable if we can find at least one linear map that separates in the sense of (3.23). The dataset in Figure 3.6 is linearly separable.

As can be verified easily, any linear map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ achieving zero average hinge loss (2.11) on the dataset $\mathcal{D}$ perfectly satisfies this dataset (3.23). It seems reasonable to learn a linear map by minimizing the average hinge loss (2.11). However, one drawback of this approach is that there might be (infinitely) many different linear maps that achieve zero average hinge loss and, in turn, perfectly separate the data points in Figure 3.6. Indeed, consider a linear map $h^{(\mathbf{w})}$ that achieves zero average hinge loss for the $\mathcal{D}$ in Figure 3.6 (and therefore perfectly separates it). Then, any other linear map $h^{(\mathbf{w}')}$ with weights $\mathbf{w}' = \lambda\mathbf{w}$, using an arbitrary number $\lambda > 1$ also achieves zero average hinge loss (and perfectly separates the dataset).

Neither the separability requirement (3.23) nor the hinge loss (2.11) are sufficient as a sole training criterion. Indeed, there are many (if not most) datasets that are not linearly separable. Even for a linearly separable dataset (such as the one Figure 3.6), there are infinitely many linear maps with zero average hinge loss. Which one of these infinitely many different maps should we use? To settle these issues, the SVM uses a "regularized" hinge loss,

$$\begin{aligned} L\left((\mathbf{x},y),h^{(\mathbf{w})}\right) &:= \max\{0,1-y\cdot h^{(\mathbf{w})}(\mathbf{x})\} + \lambda\left\|\mathbf{w}\right\|_2^2 \\ &\overset{h^{(\mathbf{w})}(\mathbf{x})=\mathbf{w}^T\mathbf{x}}{=} \max\{0,1-y\cdot \mathbf{w}^T\mathbf{x}\} + \lambda\left\|\mathbf{w}\right\|_2^2. \end{aligned} \tag{3.24}$$

The loss (3.7) augments the hinge loss (2.11) by the term $\lambda\left\|\mathbf{w}\right\|_2^2$. This term is the scaled (by $\lambda > 0$) squared Euclidean norm of the weights $\mathbf{w}$ of the linear hypothesis h used to classify data points. Adding the penalty term $\lambda\left\|\mathbf{w}\right\|_2^2$ to the hinge loss (2.11) is an instance of regularization (see Chapter 7).

The loss favours linear maps $h^{(\mathbf{w})}$ that are robust against (small) perturbations of the data points. The tuning parameter λ in (3.7) controls the strength of this regularization effect and might therefore also be referred to as a regularization parameter. We will discuss regularization on a more general level in Chapter 7.

Let us now develop a useful geometric interpretation of the linear hypothesis obtained by minimizing the loss function (3.7). According to [82, Chapter 2], a classifier $h^{(\mathbf{w}_{\text{SVM}})}$ that minimizes the average loss (3.7), maximizes the distance (margin) ξ between its decision boundary and each of the two classes $\mathcal{C}^{(y=1)}$ and $\mathcal{C}^{(y=-1)}$ (see (3.22)). The decision boundary

is given by the set of feature vectors $\mathbf{x}$ satisfying $\mathbf{w}_{\rm SVM}^T \mathbf{x} = 0$,

Making the margin as large as possible is reasonable as it ensures that the resulting classifications are robust against small perturbations of the features (see Section 7.2). As depicted in Figure 3.6, the margin between the decision boundary and the classes $\mathcal{C}_1$ and $\mathcal{C}_2$ is typically determined by few data points (such as $\mathbf{x}^{(6)}$ in Figure 3.6) which are closest to the decision boundary. These data points have minimum distance to the decision boundary and are referred to as support vectors.

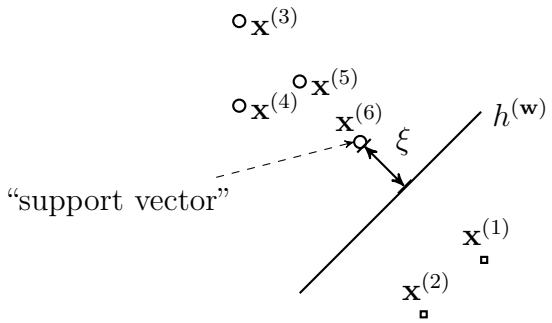


Figure 3.6: The SVM learns a hypothesis (or classifier) $h^{(\mathbf{w})}$ with minimum average soft-margin hinge loss (3.7). Minimizing this loss is equivalent to maximizing the margin ξ between the decision boundary of $h^{(\mathbf{w})}$ and each class of the training set.

We highlight that both, the SVM and logistic regression use the same hypothesis space of linear maps. Both methods learn a linear classifier $h^{(\mathbf{w})} \in \mathcal{H}^{(n)}$ (see (3.1)) whose decision boundary is a hyperplane in the feature space $\mathcal{X} = \mathbb{R}^n$ (see Figure 2.9). The difference between SVM and logistic regression is in their choice for the loss function used to evaluate the quality of a hypothesis $h^{(\mathbf{w})} \in \mathcal{H}^{(n)}$.

The hinge loss (2.11) is a (in some sense optimal) convex approximation to the 0/1 loss (2.9). Thus, we expect the classifier obtained by the SVM to yield a smaller classification error probability $p(\hat{y} \neq y)$ (with $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$ and $\hat{y} = -1$ otherwise) compared to logistic regression which uses the logistic loss (2.12). Note that using hinge loss ensures some level of robustness of the resulting ML method. Indeed, a hypothesis with maximum margin is maximally robust against perturbations of the feature vectors of data points. Section 7.2 discusses the importance of robustness in ML methods in more detail.

The statistical superiority (in terms of robustness) of the SVM comes at the cost of increased computational complexity. The hinge loss (2.11) is non-differentiable which prevents the use of simple gradient-based methods (see Chapter 5) and requires more advanced optimization methods. In contrast, the logistic loss (2.12) is convex and differentiable. Logistic

regression allows for the use of gradient-based methods to minimize the average logistic loss incurred on a training set (see Chapter 5).

3.8 Bayes Classifier

Consider data points characterized by features $\mathbf{x} \in \mathcal{X}$ and some binary label $y \in \mathcal{Y}$. We can use any two different label values but let us assume that the two possible label values are $y = -1$ or $y = 1$. We would like to find (or learn) a classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$ such that the predicted (or estimated) label $\hat{y} = h(\mathbf{x})$ agrees with the true label $y \in \mathcal{Y}$ as much as possible. Thus, it is reasonable to assess the quality of a classifier h using the 0/1 loss (2.9). We could then learn a classifier using the ERM with the loss function (2.9). However, the resulting optimization problem is typically intractable since the loss (2.9) is non-convex and non-differentiable.

Instead of solving the (intractable) ERM for 0/1 loss (2.9), we can take a different route to construct a classifier. This construction is based on a simple probabilistic model for data. Using this model, we can interpret the average 0/1 loss incurred by a hypothesis on a training set as an approximation to the probability $p_{\rm err} = p(y \neq h(\mathbf{x}))$. Any classifier $\widehat{h}$ that minimizes the error probability $p_{\rm err}$, which is the expected 0/1 loss, is referred to as a Bayes estimator. Section 4.5 will discuss ML methods using Bayes estimator in more detail.

Let us derive the Bayes estimator for a the special case of a binary classification problem. Here, data points are characterized by features $\mathbf{x}$ and label $y \in \{-1, 1\}$. Elementary probability theory allows to derive the Bayes estimator, which is the hypothesis minimizing the expected 0/1 loss, as

$$\widehat{h}(\mathbf{x}) = \begin{cases} 1 & \text{if } p(y=1|\mathbf{x}) > p(y=-1|\mathbf{x}) \\ -1 & \text{otherwise.} \end{cases} . \tag{3.25}$$

Note that the Bayes estimator (3.25) depends on the probability distribution $p(\mathbf{x}, y)$ underlying the data points.[2] We obtain different Bayes estimators for different probabilistic models. One widely used probabilistic model results in a Bayes estimator that belongs to the linear hypothesis space (3.1). Note that this hypothesis space underlies also logistic regression (see Section 3.6) and the SVM (see Section 3.7). Thus, logistic regression, SVM and Bayes estimator are all examples of a linear classifier (see Figure 2.9).

[2]Remember that we interpret data points as realizations of i.i.d. RVs with common probability distribution $p(\mathbf{x}, y)$.

A linear classifier partitions the feature space $\mathcal{X}$ into two half-spaces. One half-space consists of all feature vectors $\mathbf{x}$ which result in the predicted label $\hat{y} = 1$ and the other half-space constituted by all feature vectors $\mathbf{x}$ which result in the predicted label $\hat{y} = -1$. The family of ML methods that learn a linear classifier differ in their choices for the loss functions used to assess the quality of these half-spaces.

3.9 Kernel Methods

Consider a ML (classification or regression) problem with an underlying feature space $\mathcal{X}$. In order to predict the label $y \in \mathcal{Y}$ of a data point based on its features $\mathbf{x} \in \mathcal{X}$, we apply a predictor h selected out of some hypothesis space $\mathcal{H}$. Let us assume that the available computational infrastructure only allows us to use a linear hypothesis space $\mathcal{H}^{(n)}$ (see (3.1)).

For some applications, using a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ is not suitable since the relation between features $\mathbf{x}$ and label y might be highly non-linear. One approach to extend the capabilities of linear hypotheses is to transform the raw features of a data point before applying a linear hypothesis h.

The family of kernel methods is based on transforming the features $\mathbf{x}$ to new features $\hat{\mathbf{x}} \in \mathcal{X}'$ which belong to a (typically very) high-dimensional space $\mathcal{X}'$ [82]. It is not uncommon that, while the original feature space is a low-dimensional Euclidean space (e.g., $\mathcal{X} = \mathbb{R}^2$), the transformed feature space $\mathcal{X}'$ is an infinite-dimensional function space.

The rationale behind transforming the original features into a new (higher-dimensional) feature space $\mathcal{X}'$ is to reshape the intrinsic geometry of the feature vectors $\mathbf{x}^{(i)} \in \mathcal{X}$ such that the transformed feature vectors $\hat{\mathbf{x}}^{(i)}$ have a "simpler" geometry (see Figure 3.7).

Kernel methods are obtained by formulating ML problems (such as linear regression or logistic regression) using the transformed features $\hat{\mathbf{x}} = \phi(\mathbf{x})$. A key challenge within kernel methods is the choice of the feature map $\phi : \mathcal{X} \to \mathcal{X}'$ which maps the original feature vector $\mathbf{x}$ to a new feature vector $\hat{\mathbf{x}} = \phi(\mathbf{x})$.

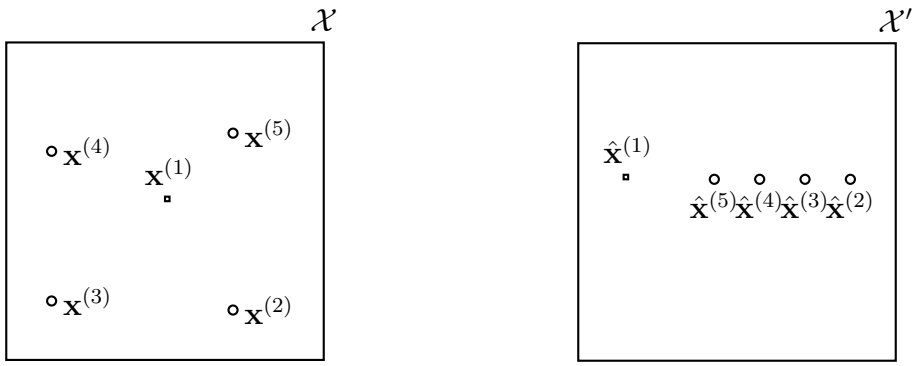


Figure 3.7: The data set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{5}$ consists of 5 data points with features $\mathbf{x}^{(i)}$ and binary labels $y^{(i)}$. Left: In the original feature space $\mathcal{X}$, the data points cannot be separated perfectly by any linear classifier Right: The feature map $\phi : \mathcal{X} \to \mathcal{X}'$ transforms the features $\mathbf{x}^{(i)}$ to the new features $\hat{\mathbf{x}}^{(i)} = \phi\big(\mathbf{x}^{(i)}\big)$ in the new feature space $\mathcal{X}'$. In the new feature space $\mathcal{X}'$ the data points can be separated perfectly by a linear classifier.

3.10 Decision Trees

A decision tree is a flowchart-like description of a map $h : \mathcal{X} \to \mathcal{Y}$ which maps the features $\mathbf{x} \in \mathcal{X}$ of a data point to a predicted label $h(\mathbf{x}) \in \mathcal{Y}$ [58]. While we can use decision trees for an arbitrary feature space $\mathcal{X}$ and label space $\mathcal{Y}$, we will discuss them for the particular feature space $\mathcal{X} = \mathbb{R}^2$ and label space $\mathcal{Y} = \mathbb{R}$.

Figure 3.8 depicts an example for a decision tree. A decision tree, consists of nodes which are connected by directed edges. We can think of a decision tree,as a step-by-step instruction, or a "recipe", for how to compute the function value $h(\mathbf{x})$ given the features $\mathbf{x} \in \mathcal{X}$ of a data point. This computation starts at the "root" node and ends at one of the "leaf" nodes of the decision tree.

A leaf node $\hat{y}$, which does not have any outgoing edges, represents a decision region $\mathcal{R}_{\hat{y}} \subseteq \mathcal{X}$ in the feature space. The hypothesis h associated with a decision tree, is constant over the regions $\mathcal{R}_{\hat{y}}$, such that $h(\mathbf{x}) = \hat{y}$ for all $\mathbf{x} \in \mathcal{R}_{\hat{y}}$ and some label value $\hat{y} \in \mathbb{R}$.

The nodes in a decision tree are of two different types,

- decision (or test) nodes, which represent particular "tests" about the feature vector $\mathbf{x}$, e.g., "is the norm of $\mathbf{x}$ larger than 10?").
- leaf nodes, which correspond to subsets of the feature space.

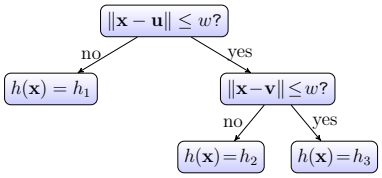
The particular decision tree,depicted in Figure 3.8 consists of two decision nodes (including

the root node) and three leaf nodes.

Given limited computational resources, we can only use decision trees with a limited depth. The depth of a decision tree, is the maximum number of hops it takes to reach a leaf node starting from the root and following the arrows. The decision tree,depicted in Figure 3.8 has depth 2. We obtain an entire hypothesis space by collecting all hypothesis maps that are obtained from the decision tree in Figure 3.8 with some vectors $\mathbf{u}$ and $\mathbf{v}$, some positive radius $w > 0$. The resulting hypothesis space is parametrized by the vectors $\mathbf{u}, \mathbf{v}$ and the number w.

To assess the quality of a particular decision tree,we can use various loss functions. Examples of loss functions used to measure the quality of a decision tree, are the squared error loss for numeric labels (regression) or the impurity of individual decision region for categorical labels (classification).

Decision tree methods use as a hypothesis space the set of all hypotheses which represented by a family of decision trees. Figure 3.9 depicts a collection of decision trees which are characterized by having depth at most two. More generally, we can construct a collection of decision trees using a fixed set of "elementary tests" on the input feature vector such as $\|\mathbf{x}\| > 3$, $x_3 < 1$. These tests might also involve a continuous (real-valued) parameter such as $\{x_2 > w\}_{w \in [0,10]}$. We then build a hypothesis space by considering all decision trees not exceeding a maximum depth and whose decision nodes carry out one of the elementary tests.



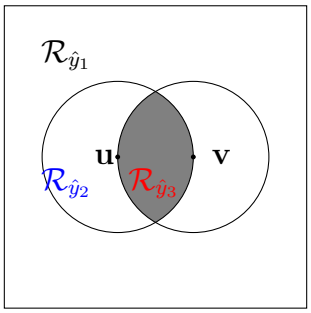


Figure 3.8: A decision tree represents a hypothesis h which is constant on the decision region $\mathcal{R}_{\hat{y}}$, i.e., $h(\mathbf{x}) = \hat{y}$ for all $\mathbf{x} \in \mathcal{R}_{\hat{y}}$. Each decision region $\mathcal{R}_{\hat{y}} \subseteq \mathcal{X}$ corresponds to a specific leaf node in the decision tree.

A decision tree represents a map $h : \mathcal{X} \to \mathcal{Y}$, which is piecewise-constant over regions of the feature space $\mathcal{X}$. These non-overlapping decision regions partition the feature space into subsets that are all mapped to the same predicted label. Each leaf node of a decision tree corresponds to one particular decision region. Using large decision trees with many test nodes, we can learn a hypothesis with many different decision regions. These decision regions

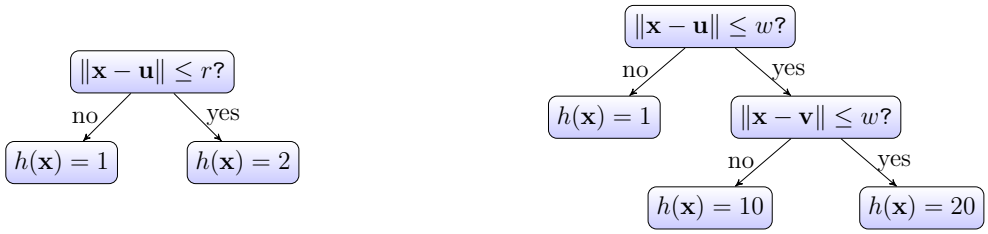


Figure 3.9: A hypothesis space $\mathcal{H}$ which consists of two decision trees with depth 2 and using the tests $\|\mathbf{x}-\mathbf{u}\| \leq w$ and $\|\mathbf{x}-\mathbf{v}\| \leq w$ with a fixed radius w and vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$.

can be chosen such they perfectly align with a given labeled dataset (see Figure 3.10).

The hypothesis space spanned by sufficiently large (deep) decision tree allows to approximate any given non-linear map (under mild technical conditions such as Lipschitz continuity). This is quite different from ML methods using the linear hypothesis space (3.1), such as linear regression, logistic regression or the SVM. These methods learn linear hypothesis maps with a rather simple geometry. Indeed, a linear map is constant along hyperplanes. Moreover, the decision regions obtained from linear classifiers are always entire half-spaces (see Figure 2.9).

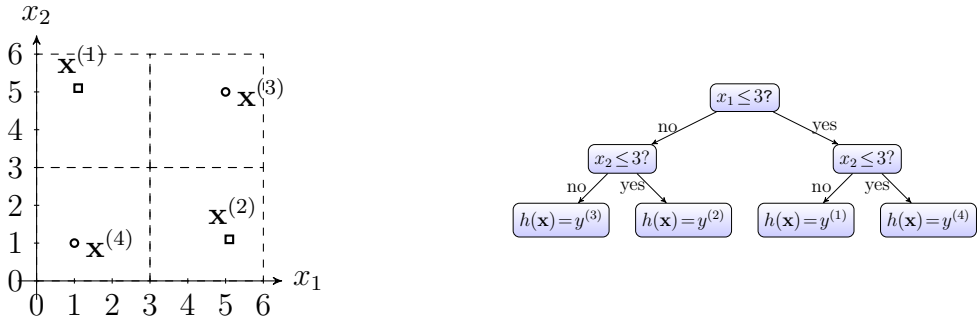


Figure 3.10: Using a sufficiently large (deep) decision tree, we can construct a map h that perfectly fits any given labeled dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ such that $h(\mathbf{x}^{(i)}) = y^{(i)}$ for $i = 1, \ldots, m$.

3.11 Deep Learning

Another example of a hypothesis space uses a signal-flow representation of a hypothesis $h : \mathbb{R}^n \to \mathbb{R}$. This signal-flow representation is referred to as a ANN. As the name indicates, an ANN is a network of interconnected elementary computational units. These computational units might be referred to as artificial neurons or just neurons.

Figure 3.11 depicts the simplest possible ANN that consists of a single neuron. The neuron computes a weighted sum of the inputs and then applies an activation function $\sigma(z)$ to produce the output $\sigma(z)$. The j-th input of a neuron is assigned a parameter or weight w_j. For a given choice of weights the ANN in Figure 3.11 represents a hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \sigma(z) = \sigma\big(\sum_j w_j x_j\big)$.

The ANN in Figure 3.11 defines a hypothesis space that is constituted by all maps $h^{(\mathbf{w})}$ obtained for different choices for the weights $\mathbf{w}$ in Figure 3.11. Note that the single-neuron ANN in Figure 3.11 reduces to a linear map when we use the activation function $\sigma(z) = z$. However, even when we use a non-linear activation function in Figure 3.11, the resulting hypothesis space is essentially the same as the space of linear maps (3.1). In particular, if we threshold the output of the ANN in Figure 3.11 to obtain a binary label, we will always obtain a linear classifier like logistic regression and SVM (see Section 3.6 and Section 3.7).

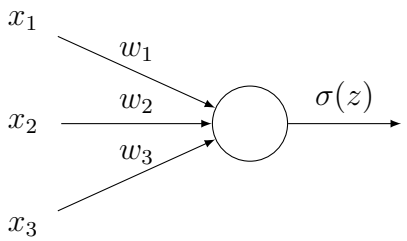


Figure 3.11: ANN consisting of a single neuron that implements a weighted summation $z = \sum_j w_j x_j$ of its inputs x_j followed by applying a non-linear activation function $\sigma(z)$.

Deep learning methods use ANN consisting of many (thousands to millions) interconnected neurons [48]. The interconnections between neurons can be arbitrary in general. However, it turns out to be beneficial to organize neurons as layers and only connect neurons in consecutive layers [48]. Figure 3.12 depicts an example for a ANN consisting of

one hidden layer to represent a (parametrized) hypothesis $h^{(\mathbf{w})} : \mathbb{R}^n \to \mathbb{R}$. The first layer

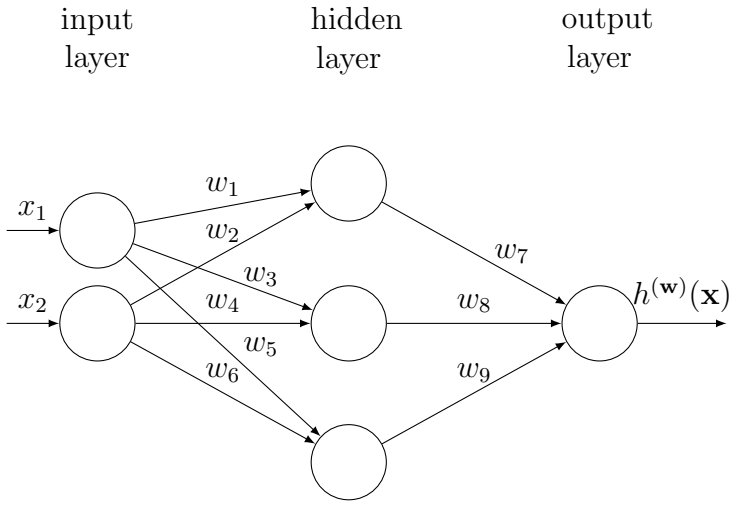


Figure 3.12: ANN representation of a hypothesis $h^{(\mathbf{w})}(\mathbf{x})$ which maps the features (input) $\mathbf{x} = (x_1, x_2)^T$ to a predicted label (output) $h^{(\mathbf{w})}(\mathbf{x})$. The depicted ANN defines a hypothesis space consisting of all maps $h^{(\mathbf{w})}(x)$ obtained from all possible choices for the weights $\mathbf{w} = (w_1, \ldots, w_9)^T$.

of the ANN in Figure 3.12 is referred to as the input layer. The input layer reads in the feature vector $\mathbf{x} \in \mathbb{R}^n$ of a data point. The features x_j are then multiplied with the weights $w_{j,j'}$ associated with the link between the jth input node ("neuron") with the j'th node in the middle (hidden) layer. The output of the j'-th node in the hidden layer is given by $s_{j'} = \sigma(\sum_{j=1}^{n} w_{j,j'} x_j)$ with some (typically non-linear) activation function $\sigma : \mathbb{R} \to \mathbb{R}$. The input to the activation function is the weighted combination $\sum_{j=1}^{n} w_{j,j'} s_{j'}$ of the outputs s_j of the nodes in the previous layer. For the ANN depicted in Figure 3.12, the output of neuron s_1 is $\sigma(z)$ with $z = w_{1,1}x_1 + w_{1,2}x_2$.

The hypothesis map represented by an ANN is parametrized by the weights of the connections between neurons. Moreover, the resulting hypothesis map depends also on the choice for the activation functions of the individual neurons. These activation function are a design choice that can be adapted to the statistical properties of the data. However, a few particular choices for the activation function have proven useful in many important application domains. Two popular choices for the activation function used within ANNs are the sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$ or the rectified linear unit (ReLU) $\sigma(z) = \max\{0, z\}$. ANNs using many, say 10, hidden layers, is often referred to as a "deep net". ML methods

using hypothesis spaces obtained from deep ANN (deep net)s are known as deep learning methods [48].

It can be shown that an ANN with only one single (but arbitrarily large) hidden layer can approximate any given map $h : \mathcal{X} \rightarrow \mathcal{Y} = \mathbb{R}$ to any desired accuracy [28]. Deep learning methods often use a ANN with a relatively large number (more than hundreds) of hidden layers. We refer to a ANN with a relatively large number of hidden layers as a deep net.

There is empirical and theoretical evidence that using many hidden layers, instead of few but wide layers, is computationally and statistically favourable [33, 114] and [48, Ch. 6.4.1.]. The hypothesis map $h^{(\mathbf{w})}$ represented by an ANN can be evaluated (to obtain the predicted label at the output) efficiently using message passing over the ANN. This message passing can be implemented using parallel and distributed computers. Moreover, the graphical representation of a parametrized hypothesis in the form of a ANN allows us to efficiently compute the gradient of the loss function via a (highly scalable) message passing procedure known as back-propagation [48]. Being able to quickly compute gradients is instrumental for the efficiency of gradient based methods for learning a good choice for the ANN weights (see Chapter 5).

3.12 Maximum Likelihood

For many applications it is useful to model the observed data points $\mathbf{z}^{(i)}$, with $i = 1, \ldots, m$, as i.i.d. realizations of a RV $\mathbf{z}$ with probability distribution $p(\mathbf{z}; \mathbf{w})$. This probability distribution is parametrized by a parameter vector $\mathbf{w} \in \mathbb{R}^n$. A principled approach to estimating the parameter vector $\mathbf{w}$ based on a set of i.i.d. realizations $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)} \sim p(\mathbf{z}; \mathbf{w})$ is maximum likelihood estimation [85].

Maximum likelihood estimation can be interpreted as an ML problem with a hypothesis space parametrized by the parameter vector $\mathbf{w}$. Each element $h^{(\mathbf{w})}$ of the hypothesis space $\mathcal{H}$ corresponds to one particular choice for the parameter vector $\mathbf{w}$. Maximum likelihood methods use the loss function

$$L\left(\mathbf{z}, h^{(\mathbf{w})}\right) := -\log p(\mathbf{z}; \mathbf{w}). \tag{3.26}$$

A widely used choice for the probability distribution $p\big(\mathbf{z}; \mathbf{w}\big)$ is a multivariate normal (Gaussian) distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, both of which constitute the parameter vector $\mathbf{w} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ (we have to reshape the matrix $\boldsymbol{\Sigma}$ suitably into a vector form).

Given the i.i.d. realizations $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)} \sim p\left(\mathbf{z}; \mathbf{w}\right)$, the maximum likelihood estimates $\hat{\boldsymbol{\mu}}$, $\widehat{\boldsymbol{\Sigma}}$ of the mean vector and the covariance matrix are obtained via

$$\hat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}} = \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathbb{R}^n, \boldsymbol{\Sigma} \in \mathbb{S}^n_+} (1/m) \sum_{i=1}^{m} - \log p\big(\mathbf{z}^{(i)}; (\boldsymbol{\mu}, \boldsymbol{\Sigma})\big). \tag{3.27}$$

The optimization in (3.27) is over all possible choices for the mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{S}^n_+$. Here, $\mathbb{S}^n_+$ denotes the set of all psd Hermitian $n \times n$ matrices.

The maximum likelihood problem (3.27) can be interpreted as an instance of ERM (4.3) using the particular loss function (3.26). The resulting estimates are given explicitly as

$$\hat{\boldsymbol{\mu}} = (1/m) \sum_{i=1}^{m} \mathbf{z}^{(i)}, \text{ and } \widehat{\boldsymbol{\Sigma}} = (1/m) \sum_{i=1}^{m} (\mathbf{z}^{(i)} - \hat{\boldsymbol{\mu}})(\mathbf{z}^{(i)} - \hat{\boldsymbol{\mu}})^T. \tag{3.28}$$

Note that the expressions (3.28) are only valid when the probability distribution of the data points is modelled as a multivariate normal distribution.

3.13 Nearest Neighbour Methods

Nearest neighbour (NN) methods are a family of ML methods that are characterized by a specific construction of the hypothesis space. NN methods can be applied to regression problems involving numeric labels (e.g., using label space $\mathcal{Y} = \mathbb{R}$) as well as for classification problems involving categorical labels (e.g., with label space $\mathcal{Y} = \{-1, 1\}$).

While NN methods can be combined with arbitrary label spaces, they require the feature space to have a specific structure. NN methods require the feature space be a metric space [119] that provides a measure for the distance between different feature vectors. We need a metric or distance measure to determine the nearest neighbour of a data point. A prominent example for a metric feature space is the Euclidean space $\mathbb{R}^n$. The metric of $\mathbb{R}^n$ given by the Euclidean distance $\|\mathbf{x} - \mathbf{x}'\|$ between two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$.

Consider a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$ that consists of labeled data points. Thus, for each data point we know the features and the label value. Given such a training set, NN methods construct a hypothesis space that consist of piece-wise constant maps $h : \mathcal{X} \to \mathcal{Y}$. For any hypothesis h in that space, the function value $h(\mathbf{x})$ depends only on the (labels of the) k nearest data points (smallest distance to $\mathbf{x}$) in the training set $\mathcal{D}$. The number k of NNs used to determine the function value $h(\mathbf{x})$ is a hyper-parameter of the NN method. NN methods are also referred to as k-NN methods to make their dependence on the parameter

k explicit.

Let us illustrate NN methods by considering a binary classification problem using an uneven number for k (e.g., $k = 3$ or $k = 5$). The goal is to learn a hypothesis that predicts the binary label $y \in \{-1, 1\}$ of a data point based on its feature vector $\mathbf{x} \in \mathbb{R}^n$. This learning task can make use of a training set $\mathcal{D}$ containing $m > k$ data points with known labels. Given a data point with features $\mathbf{x}$, denote by $\mathcal{N}^{(k)}$ a set of k data points in $\mathcal{D}$ whose feature vectors have smallest distance to $\mathbf{x}$. The number of data points in $\mathcal{N}^{(k)}$ whose label is 1 is denoted $m_1^{(k)}$ and those with label value -1 is denoted $m_{-1}^{(k)}$. The k-NN method "learns" a hypothesis $\widehat{h}$ given by

$$\widehat{h}(\mathbf{x}) = \begin{cases} 1 & \text{if } m_1^{(k)} > m_{-1}^{(k)} \\ -1 & \text{otherwise.} \end{cases} \tag{3.29}$$

It is important to note that, in contrast to the ML methods in Section 3.1 - Section 3.11, the hypothesis space of k-NN depends on a labeled dataset (training set) $\mathcal{D}$. As a consequence, k-NN methods need to access (and store) the training set whenever the compute a prediction (evaluate $h(\mathbf{x})$). To compute the prediction $h(\mathbf{x})$ for a data point with features $\mathbf{x}$, k-NN needs to determine its NNs in the training set. When using a large training set this implies a large storage requirement for k-NN methods. Moreover, k-NN methods might be prone to revealing sensitive information with its predictions (see Exercise 3.9).

For a fixed k, NN methods do not require any parameter tuning. Such parameter tuning (or learning) is required linear regression, logistic regression and deep learning methods. In contrast, the hypothesis "learnt" by NN methods is characterized point-wise, for each possible value of features $\mathbf{x}$, by the NN in the training set. Compared to the ML methods in Section 3.1 - Section 3.11, NN methods do not require to solve (challenging) optimization problems for model parameters. Beside their low computational requirements (put aside the memory requirements), NN methods are also conceptually appealing as natural approximations of Bayes estimators (see [26] and Exercise 3.10).

3.14 Deep Reinforcement Learning

Deep reinforcement learning (DRL) refers to a subset of ML problems and methods that revolve around the control of dynamic systems such as autonomous driving cars or cleaning robots [129, 137, 104]. A DRL problem involves data points that represent the states of a dynamic system at different time instants $t = 0, 1, \ldots$. The data points representing the state at some time instant t is characterized by the feature vector $\mathbf{x}^{(t)}$. The entries of this

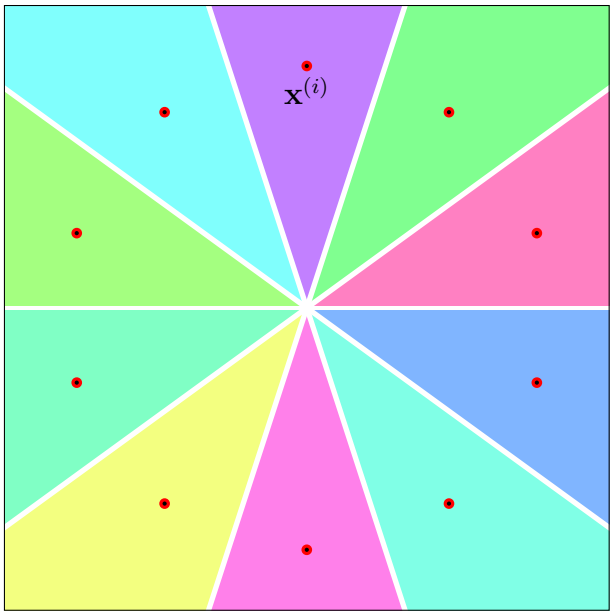


Figure 3.13: A hypothesis map h for k-NN with $k = 1$ and feature space $\mathcal{X} = \mathbb{R}^2$. The hypothesis map is constant over regions (indicated by the coloured areas) located around feature vectors $\mathbf{x}^{(i)}$ (indicated by a dot) of some data $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}$.

feature vector are the individual features of the state at time t. These features might be obtained via sensors, onboard-cameras or other ML methods (that predict the location of the dynamic system). The label $y^{(t)}$ of a data point might represent the optimal steering angle at time t.

DRL methods learn a hypothesis h that delivers optimal predictions $\hat{y}^{(t)} := h\big(\mathbf{x}^{(t)}\big)$ for the optimal steering angle $y^{(t)}$. As their name indicates, DRL methods use hypothesis spaces obtained from a deep net (see Section 3.11). The quality of the prediction $\hat{y}^{(t)}$ obtained from a hypothesis is measured by the loss $L\left(\big(\mathbf{x}^{(t)}, y^{(t)}\big), h\right) := -r^{(t)}$ with a reward signal $r^{(t)}$. This reward signal might be obtained from a distance (collision avoidance) sensor or low-level characteristics of an on-board camera snapshot.

The (negative) reward signal $-r^{(t)}$ typically depends on the feature vector $\mathbf{x}^{(t)}$ and the discrepancy between optimal steering direction $y^{(t)}$ (which is unknown) and its prediction $\hat{y}^{(t)} := h\big(\mathbf{x}^{(t)}\big)$. However, what sets DRL methods apart from other ML methods such as linear regression (see Section 3.1) or logistic regression (see Section 3.6) is that they can evaluate the loss function only point-wise $L\left(\big(\mathbf{x}^{(t)}, y^{(t)}\big), h\right)$ for the specific hypothesis h that has been used to compute the prediction $\hat{y}^{(t)} := h\big(\mathbf{x}^{(t)}\big)$ at time instant t. This is fundamentally different from linear regression that uses the squared error loss (2.8) which can be evaluated for every possible hypothesis $h \in \mathcal{H}$.

3.15 LinUCB

ML methods are instrumental for various recommender systems [86]. A basic form of a recommender system amount to chose at some time instant t the most suitable item (product, song, movie) among a finite set of alternatives $a = 1, \ldots, A$. Each alternative is characterized by a feature vector $\mathbf{x}^{(t,a)}$ that varies between different time instants.

The data points arising in recommender systems might represent time instants t at which recommendations are computed. The data point at time t is characterized by a feature vector

$$\mathbf{x}^{(t)} = \big(\big(\mathbf{x}^{(t,1)}\big)^T, \ldots, \big(\mathbf{x}^{(t,A)}\big)^T \big)^T. \tag{3.30}$$

The feature vector $\mathbf{x}^{(t)}$ is obtained by stacking the feature vectors of alternatives at time t into a single long feature vector. The label of the data point t is a vector of rewards $\mathbf{y}^{(t)} := \big(r_1^{(t)}, \ldots, r_A^{(t)}\big)^T \in \mathbb{R}^A$. The entry $r_a^{(t)}$ represents the reward obtained by choosing (recommending) alternative a (with features $\mathbf{x}^{(t,a)}$) at time t. We might interpret the reward $r^{(t,a)}$ as an indicator if the costumer actually buys the product corresponding to the recommended alternative a.

The ML method LinUCB (the name seems to be inspired by the terms "linear" and "upper confidence bound" (UCB)) aims at learning a hypothesis h that allows to predict the rewards $\mathbf{y}^{(i)}$ based on the feature vector $\mathbf{x}^{(t)}$ (3.30). As its hypothesis space $\mathcal{H}$, LinUCB uses the space of linear maps from the stacked feature vectors $\mathbb{R}^{nA}$ to the space of reward vectors $\mathbb{R}^A$. This hypothesis space can be parametrized by matrices $\mathbf{W} \in \mathbb{R}^{A \times nA}$. Thus, LinUCB learns a hypothesis that computes predicted rewards via

$$\widehat{\mathbf{y}}^{(t)} := \mathbf{W}\mathbf{x}^{(t)}. \tag{3.31}$$

The entries of $\widehat{\mathbf{y}}^{(t)} = \big(\hat{r}_1^{(t)}, \ldots, \hat{r}_A^{(t)}\big)$ are predictions of the individual rewards $r^{(t,a)}$. It seems natural to recommend at time t the alternative a whose predicted reward is maximum. However, it turns out that this approach is sub-optimal as it prevents the recommender system from learning the optimal predictor map $\mathbf{W}$.

Loosely speaking, LinUCB tries out (explores) each alternative $a \in \{1, \ldots, A\}$ sufficiently often to obtain a sufficient amount of training data for learning a good weight matrix $\mathbf{W}$. At time t, LinUCB chooses the alternative $a^{(t)}$ that maximizes the quantity

$$\hat{r}_a^{(t)} + R(t,a) \ , \ a = 1, \ldots, A. \tag{3.32}$$

We can think of the component $R(t,a)$ as a measure for the uncertainty in the reward obtained by action a. It is constructed such that (3.32) upper bounds the actual reward $r_a^{(t)}$ with a prescribed level of confidence (or probability). The confidence term $R(t,a)$ depends on the feature vectors $\mathbf{x}^{(t',a)}$ of the alternative a at previous time instants $t' < t$. Thus, at each time instant t, LinUCB chooses the alternative a that results in the largest upper confidence bound (UCB) (3.32) on the reward (hence the "UCB" in LinUCB). We refer to the relevant literature on sequential learning and decision-making for more details on the LinUCB method [86].

3.16 Exercises

Exercise 3.1. Logistic loss and Accuracy. Section 3.6 discussed logistic regression as a ML method that learns a linear hypothesis map by minimizing the logistic loss (3.16). The logistic loss has computationally pleasant properties as it is smooth and convex. However, in some applications we might be ultimately interested in the accuracy or (equivalently) the average 0/1 loss (2.9). Can we upper bound the average 0/1 loss using the average logistic loss incurred by a given hypothesis on a given training set?

Exercise 3.2. Robustness Against Outliers. Consider the dataset

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

The i-th data point is characterized by the feature $x^{(i)} = i/m$ and label $y^{(i)} = i/m$. Linear regression learns a linear hypothesis $h^{(\mathbf{w})}(x) = w_1 x + w_2$, with parameter vector $\mathbf{w} = \big(w_1, w_2\big)^T$ such that the average squared error loss $(1/m)\sum_{i=1}^{m}\big(y^{(i)} - h(x^{(i)})\big)^2$ is minimized. Let us denote the resulting (learnt) parameter vector as $\widehat{\mathbf{w}} = \big(\hat{w}_1, \hat{w}_2\big)^T$. Consider now that the original dataset is perturbed, resulting in a new dataset $\widetilde{\mathcal{D}}$. The dataset $\widetilde{\mathcal{D}}$ is obtained from $\mathcal{D}$ by replacing the first data point $\big(1/m, 1/m\big)$ with an outlier with feature value $x^{(o)} = c$ and label value $y^{(o)} = 0$. We then apply linear regression to this perturbed dataset which results in a parameter vector $\widehat{\mathbf{w}}^{(o)} := \big(\hat{w}_1^{(o)}, \hat{w}_2^{(o)}\big)^T$. Discuss the deviation between the parameter vectors $\widehat{\mathbf{w}}$ and $\widehat{\mathbf{w}}^{(o)}$ as a function of the outlier feature value c.

Exercise 3.3. How Many Neurons? Consider a predictor map $h(x)$ which is piecewise linear and consisting of 1000 pieces. Assume we want to represent this map by an ANN using neurons with one hidden layer of neurons having a ReLU activation function. The

output layer consists of a single neuron with linear activation function. How many neurons must the ANN contain at least ?

Exercise 3.4. Effective Dimension of ANN. Consider a ANN with $n = 10$ input neurons following by three hidden layers consisting of 4, 9 and 3 nodes. The three hidden layers are followed by the output layer consisting of a single neuron. Assume that all neurons use a linear activation function and no bias term. What is the effective dimension $d_{\text{eff}}(\mathcal{H})$ of the hypothesis space $\mathcal{H}$ that consists of all hypothesis maps that can be obtained from this ANN.

Exercise 3.5. Linear Classifiers. Consider data points characterized by feature vectors $\mathbf{x} \in \mathbb{R}^n$ and binary labels $y \in \{-1, 1\}$. We are interested in finding a good linear classifier which is such that the feature vectors resulting in $h(\mathbf{x}) = 1$ is a half-space. Which of the methods discussed in this chapter aim at learning a linear classifier?

Exercise 3.6. Data Dependent Hypothesis space. Consider a ML application involving data points with features $\mathbf{x} \in \mathbb{R}^6$ and a numeric label $y \in \mathbb{R}$. We learn a hypothesis by minimizing the average loss incurred on a training set $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$. Which of the following ML methods uses a hypothesis space that depends on the dataset $\mathcal{D}$?

- logistic regression
- linear regression
- k-NN

Exercise 3.7. Triangle. Consider the ANN in Figure 3.12 using the ReLU activation function (see Figure 3.11). Show that there is a particular choice for the weights $\mathbf{w} = (w_1, \ldots, w_9)^T$ such that the resulting hypothesis map $h^{(\mathbf{w})}(x)$ is a triangle as depicted in Figure 3.14. Can you also find a choice for the weights $\mathbf{w} = (w_1, \ldots, w_9)^T$ that produce the same triangle shape if we replace the ReLU activation function with the linear function $\sigma(z) = 10 \cdot z$?

Exercise 3.8. Approximate Triangle with Gaussians Try to approximate the hypothesis map depicted in Figure 3.14 by an element of $\mathcal{H}_{\text{Gauss}}$ (see (3.15)) using $\sigma = 1/10$, $n = 10$ and $\mu_j = -1 + (2j/10)$.

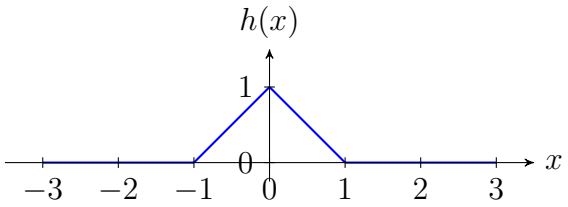


Figure 3.14: A hypothesis map $h : \mathbb{R} \to \mathbb{R}$ with the shape of a triangle.

Exercise 3.9. Privacy Leakage in k-NN Consider a k-NN method for a binary classification problem. We use $k = 1$ and a given training set whose data points characterize humans. Each human is characterized by a feature vector and label that indicates sensitive information (e.g., some sickness). Assume that you have access to the feature vectors of the data points in the training set but not to their labels. Can you infer the label value of a data point in the training set based on the prediction that you obtained based on your feature vector?

Exercise 3.10. k-NN Approximates Bayes estimator Consider a binary classification problem involving data points that are characterized by feature vectors $\mathbf{x} \in \mathbb{R}^n$ and binary labels $y \in \{-1, 1\}$. We have access to a labeled training set $\mathcal{D}$ of size m. Show that the k-NN hypothesis (3.29) is obtained from the Bayes estimator (3.25) by approximating or estimating the conditional probability distribution $p(\mathbf{x}|y)$ via the density estimator [12, Sec. 2.5.2.]

$$\hat{p}(\mathbf{x}\ |y) := (k/m)\frac{1}{\mathrm{vol}(R_k)}. \tag{3.33}$$

Here, $\mathrm{vol}(R)$ denotes the volume of a ball with radius R and R_k is the distance between $\mathbf{x}$ and the kth nearest feature vector of a data point in $\mathcal{D}$.

Chapter 4

Empirical Risk Minimization

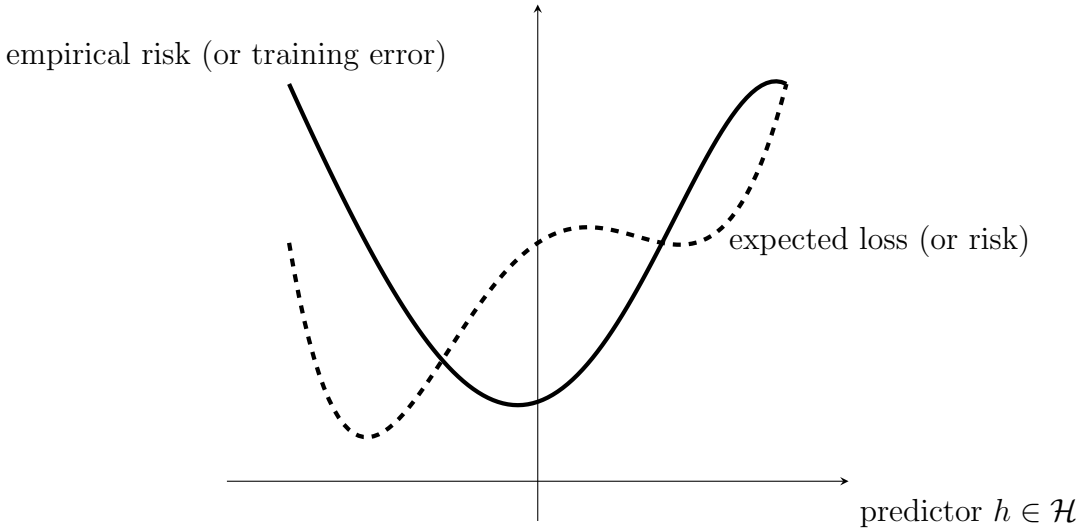


Figure 4.1: ML methods learn a hypothesis $h \in \mathcal{H}$ that incur small loss when predicting the label y of a data point based on its features $\mathbf{x}$. ERM approximates the expected loss or risk by the empirical risk (solid curve) incurred on a finite set of labeled data points (the training set). Note that we can compute the empirical risk based on the observed data points. However, to compute the risk we would need to know the underlying probability distribution which is rarely the case.

Chapter 2 discussed three components (see Figure 2.1):

- data points characterized by features $\mathbf{x} \in \mathcal{X}$ and labels $y \in \mathcal{Y}$,
- a hypothesis space $\mathcal{H}$ of computationally feasible maps $h : \mathcal{X} \to \mathcal{Y}$,

- and a loss function $L\left((\mathbf{x}, y), h\right)$ that measures the discrepancy between the predicted label $h(\mathbf{x})$ and the true label y.

Ideally we would like to learn a hypothesis $h \in \mathcal{H}$ such that $L\left((\mathbf{x}, y), h\right)$ is small for any data point $(\mathbf{x}, y)$. However, to implement this informal goal we need to define what is meant precisely by "any data point". Maybe the most widely used approach to the define the concept of "any data point" is the i.i.d. assumption.

The i.i.d. assumption interprets data points as realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{x}, y)$. The probability distribution $p(\mathbf{x}, y)$ allows us to define the risk of a hypothesis h as the expectation of the loss incurred by h on (the realizations of) a random data point. We can interpret the risk of a hypothesis as a measure for its quality in predicting the label of"any data point".

If we know the probability distribution $p(\mathbf{x}, y)$ from which data points are drawn (i.i.d.), we can precisely determine the hypothesis with minimum risk. This optimal hypothesis, which is referred to as a Bayes estimator, can be read off almost directly from the posterior probability distribution $p(y|\mathbf{x})$ of the label y given the features $\mathbf{x}$ of a data point. The precise form of the Bayes estimator depends on the choice for the loss function. When using the squared error loss, the optimal hypothesis (or Bayes estimator) is given by the posterior mean $h(\mathbf{x}) = \mathbb{E}\big\{y|\mathbf{x}\big\}$ [85].

In most ML application, we do not know the true underlying probability distribution $p(\mathbf{x}, y)$ from which data points are generated. Therefore, we cannot compute the Bayes estimator exactly. However, we can approximately compute this estimator by replacing the exact probability distribution with an estimate or approximation. Section 4.5 discusses a specific ML method that implements this approach.

The risk of the Bayes estimator (which is the Bayes risk) provides a useful baseline against which we can compare the average loss incurred by a ML method on a set of data points. Section 6.6 shows how to diagnose ML methods by comparing its average loss of a hypothesis on a training set and its average loss on a validation set with a baseline.

Section (4.1) motivates ERM by approximating the risk using the empirical risk (or average loss) computed for a set of labeled (training) data points (see Figure 4.1). This approximation is justified by the law of large numbers which characterizes the deviation between averages of RVs and their expectation. Section 4.2 discusses the statistical and computational aspects of ERM. We then specialize the ERM for three particular ML methods arising from different combinations of hypothesis space and loss function. Section 4.3 discusses ERM for linear regression (see Section 3.1). Here, ERM amounts to minimizing a

differentiable convex function, which can be done efficiently using gradient-based methods (see Chapter 5).

We then discuss in Section 4.4 the ERM obtained for decision tree models. The resulting ERM problems becomes a discrete optimization problem which are typically much harder than convex optimization problems. We cannot apply gradient-based methods to solve the ERM for decision trees. To solve ERM for a decision tree, we essentially must try out all possible choices for the tree structure [64].

Section 4.5 considers the ERM obtained when learning a linear hypothesis using the 0/1 loss for classification problems. The resulting ERM amounts to minimizing a non-differentiable and non-convex function. Instead of applying optimization methods to solve this ERM instance, we will instead directly construct approximations of the Bayes estimator.

Section 4.6 decomposes the operation of ML methods into training periods and inference periods. The training period amounts to learning a hypothesis by solving the ERM on a given training set. The resulting hypothesis is then applied to new data points, which are not contained in the training set. This application of a learnt hypothesis to data points outside the training set is referred to as the inference period. Section 4.7 demonstrates how an online learning method can be obtained by solving the ERM sequentially as new data points come in. Online learning methods alternate between training and inference periods whenever new data is collected.

4.1 Approximating Risk by Empirical Risk

The data points arising in many important application domains can be modelled (or approximated) as realizations of i.i.d. RVs with a common (joint) probability distribution $p(\mathbf{x}, y)$ for the features $\mathbf{x}$ and label y. The probability distribution $p(\mathbf{x}, y)$ used in this i.i.d. assumption allows us to define the expected loss or risk of a hypothesis $h \in \mathcal{H}$ as

$$\mathbb{E}\big\{ L\left((\mathbf{x}, y), h\right)\big\}. \tag{4.1}$$

It seems reasonable to learn a hypothesis h such that its risk (4.1) is minimal,

$$h^* := \operatorname*{argmin}_{h \in \mathcal{H}} \mathbb{E}\big\{ L\left((\mathbf{x}, y), h\right)\big\}. \tag{4.2}$$

We refer to any hypothesis h^* that achieves the minimum risk (4.2) as a Bayes estimator [85]. Note that the Bayes estimator h^* depends on both, the probability distribution $p(\mathbf{x}, y)$

and the loss function. When using the squared error loss (2.8) in (4.2), the Bayes estimator h^* is given by the posterior mean of y given the features $\mathbf{x}$ (see [109, Ch. 7]).

Risk minimization (4.2) cannot be used for the design of ML methods whenever we do not know the probability distribution $p(\mathbf{x}, y)$. If we do not know the probability distribution $p(\mathbf{x}, y)$, which is the rule for many ML applications, we cannot evaluate the expectation in (4.1). One exception to this rule is if the data points are synthetically generated by drawing realizations from a given probability distribution $p(\mathbf{x}, y)$.

The idea of ERM is to approximate the expectation in (4.2) with an average loss (the empirical risk) incurred on a given set of data points (the "training set"),

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

As discussed in Section 2.3.4, this approximation is justified by the law of large numbers. We obtain ERM by replacing the risk in the minimization problem (4.2) with the empirical risk (2.16),

$$\begin{aligned} \widehat{h} &\in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}(h|\mathcal{D}) \\ &\overset{(2.16)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right). \end{aligned} \tag{4.3}$$

As the notation in (4.3) indicates there might be several different hypotheses that minimize $\widehat{L}(h|\mathcal{D})$. We denote by $\widehat{h}$ any of them. Mathematically, ERM (4.3) is just an instance of an optimization problem [14]. The optimization domain in (4.3) is the hypothesis space $\mathcal{H}$ of a ML method, the objective function (or cost function) is the empirical risk (2.16). ML methods that learn a hypothesis via ERM (4.3) are instances of optimization algorithms [133].

ERM (4.3) is a form of "learning by trial and error". A (hypothetical) instructor (or supervisor) provides us the labels $y^{(i)}$ for the data points in $\mathcal{D}$ which are characterized by features $\mathbf{x}^{(i)}$. This dataset serves as a training set in the following sense. We use a current guess for a good hypothesis h to predict the labels $y^{(i)}$ of the data points in $\mathcal{D}$ only from their features $\mathbf{x}^{(i)}$. We then determine average loss $\widehat{L}(h|\mathcal{D})$ that is incurred by the predictions $\hat{y}^{(i)} = h\big(\mathbf{x}^{(i)}\big)$. If the training error $\widehat{L}(h|\mathcal{D})$ is too large, we try out another hypothesis map h' different from h with the hope of achieving a smaller training error $\widehat{L}(h'|\mathcal{D})$.

We highlight that ERM (4.3) is motivated by the law of large numbers. The law of large numbers, in turn, is only useful if the data points generated within an ML application can be

well modelled as realizations of i.i.d. RVs. This i.i.d. assumption is one of the most widely used working assumptions for the design and analysis of ML methods. However, there are many important application domains involving data points that clearly violate this i.i.d. assumption.

One example for non-i.i.d. data is time series data that consists of temporally ordered (consecutive) data points [15, 88]. Each data point in a time series might represent a specific time interval or single time instants. Another example for non-i.i.d. data arises in active learning where ML methods actively choose (or query) new data points [25]. For a third example of non-i.i.d. data, we refer to FL applications that involve collections (networks) of data generators with different statistical properties [93, 70, 74, 142, 124]. A detailed discussion of ML methods for non-i.i.d. data is beyond the scope of this book.

4.2 Computational and Statistical Aspects of ERM

Solving the optimization problem (4.3) provides two things. First, the minimizer $\widehat{h}$ is a predictor which performs optimal on the training set $\mathcal{D}$. Second, the corresponding objective value $\widehat{L}(\widehat{h}|\mathcal{D})$ (the "training error") can be used to estimate for the risk or expected loss of $\widehat{h}$. However, as we will discuss in Chapter 7, for some datasets $\mathcal{D}$, the training error $\widehat{L}(\widehat{h}|\mathcal{D})$ obtained for $\mathcal{D}$ can be very different from the expected loss (risk) of $\widehat{h}$ when applied to new data points which are not contained in $\mathcal{D}$.

The i.i.d. assumption implies that the training error $\widehat{L}(h|\mathcal{D})$ is only a noisy approximation of the risk $\mathbb{E}\big\{L\left((\mathbf{x},y),h\right)\big\}$. The ERM solution $\widehat{h}$ is a minimizer of this noisy approximation and therefore in general different from the Bayes estimator which minimizes the risk itself. Even if the hypothesis $\widehat{h}$ delivered by ERM (4.3) has small training error $\widehat{L}(\widehat{h}|\mathcal{D})$, it might have unacceptably large risk $\mathbb{E}\big\{L\left((\mathbf{x},y),\widehat{h}\right)\big\}$. We refer to such a situation as overfitting and will discuss techniques for detecting and avoiding it in Chapter 6.

Many important ML methods use hypotheses that are parametrized by a parameter vector $\mathbf{w}$. For each possible parameter vector, we obtain a hypothesis $h^{(\mathbf{w})}(\mathbf{x})$. Such a parametrization is used in linear regression methods which learn a linear hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with some parameter vector $\mathbf{w}$. Another example for such a parametrization is obtained from ANNs with the weights assigned to inputs of individual (artificial) neurons (see Figure 3.12).

For ML methods that use a parametrized hypothesis $h^{(\mathbf{w})}(\mathbf{x})$, we can reformulate the

optimization problem (4.3) as an optimization of the parameter vector,

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}) \text{ with } f(\mathbf{w}) := \underbrace{(1/m)\sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right)}_{\widehat{L}\left(h^{(\mathbf{w})}|\mathcal{D}\right)}. \tag{4.4}$$

The objective function $f(\mathbf{w})$ in (4.4) is the empirical risk $\widehat{L}\left(h^{(\mathbf{w})}|\mathcal{D}\right)$ incurred by the hypothesis $h^{(\mathbf{w})}$ when applied to the data points in the dataset $\mathcal{D}$. The optimization problems (4.4) and (4.3) are fully equivalent. Given the optimal parameter vector $\widehat{\mathbf{w}}$ solving (4.4), the hypothesis $h^{(\widehat{\mathbf{w}})}$ solves (4.3).

We highlight that the precise shape of the objective function $f(\mathbf{w})$ in (4.4) depends on the parametrization of the hypothesis space $\mathcal{H}$. The parametrization is the precise rule that assigns a hypothesis map $h^{(\mathbf{w})}$ to a given parameter vector $\mathbf{w}$. The shape of $f(\mathbf{w})$ depends also on the choice for the loss function $L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right)$. As depicted in Figure 4.2, the different combinations of parametrized hypothesis space and loss function can result in objective functions with fundamentally different properties such that their optimization is more or less difficult.

The objective function $f(\mathbf{w})$ for the ERM obtained for linear regression (see Section 3.1) is differentiable and convex and can therefore be minimized using simple gradient-based methods (see Chapter 5). In contrast, the objective function $f(\mathbf{w})$ of ERM obtained for least absolute deviation regression or the SVM (see Section 3.3 and 3.7) is non-differentiable but still convex. The minimization of such functions is more challenging but still tractable as there exist efficient convex optimization methods which do not require differentiability of the objective function [110].

The objective function $f(\mathbf{w})$ obtained for ANN are typically highly non-convex with many local minima (see Figure 4.2). The optimization of non-convex objective function is in general more difficult than optimizing a convex objective function (see Fig. 4.2). However, it turns out that gradient-based methods can still be successfully applied to instances of ERM that result in a non-convex objective function [48]. The implementation of ERM might be even more challenging for ML methods that use decision trees or the 0/1 loss (2.9). Indeed, the ERM instances obtained when using decision trees or the 0/1 loss (2.9)involve non-differentiable objective functions which are even harder to minimize compared to non-convex smooth functions (see Section 4.4).

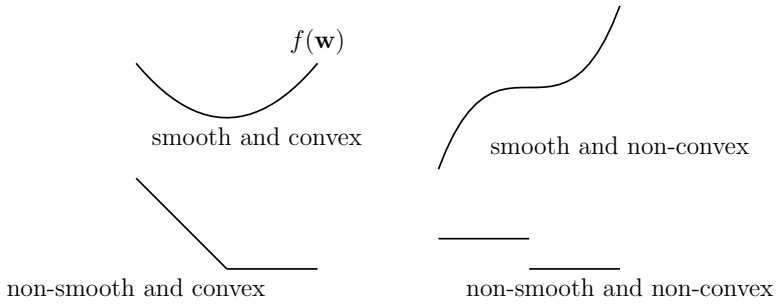


Figure 4.2: Different types of objective function in ERM arising from different combinations of hypothesis space and loss function.

4.3 ERM for Linear Regression

As discussed in Section 3.1, linear regression learns a linear hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with minimum squared error loss (2.8). For linear regression, the ERM problem (4.4) becomes

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} (1/m) \sum_{i=1}^{m} \left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2. \tag{4.5}$$

Here, $m = |\mathcal{D}|$ denotes the sample size of the training set $\mathcal{D}$. The objective function $f(\mathbf{w})$ in (4.5) is convex and smooth. Such a function can be minimized using the gradient-based methods discussed in Chapter 5. Variants of these gradient-based methods are widely used in current ML methods to solve the underlying optimization problems [48, Ch. 8].

We can rewrite the ERM problem (4.5) more concisely by stacking the labels $y^{(i)}$ and feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, into a "label vector" $\mathbf{y}$ and "feature matrix" $\mathbf{X}$,

$$\begin{aligned} \mathbf{y} &= (y^{(1)}, \ldots, y^{(m)})^T \in \mathbb{R}^m, \text{ and} \\ \mathbf{X} &= \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\right)^T \in \mathbb{R}^{m\times n}. \end{aligned} \tag{4.6}$$

This allows us to rewrite the objective function in (4.5) as

$$(1/m) \sum_{i=1}^{m} \left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2 = (1/m)\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2. \tag{4.7}$$

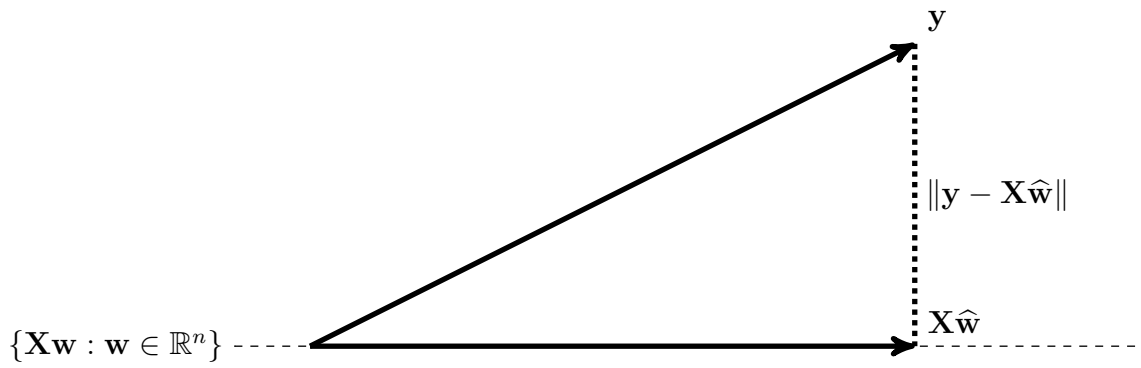


Figure 4.3: The ERM (4.8) for linear regression amounts to an orthogonal projection of the label vector $\mathbf{y} = \big(y^{(1)}, \ldots, y^{(m)}\big)^T$ on the subspace spanned by the columns of the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T$.

Inserting (4.7) into (4.5), allows to rewrite the ERM problem for linear regression as

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n} (1/m) \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2. \tag{4.8}$$

The formulation (4.8) allows for an interesting geometric interpretation of linear regression. Solving (4.8) amounts to finding a vector $\mathbf{X}\mathbf{w}$ (with feature matrix $\mathbf{X}$ (4.6)), that is closest (in the Euclidean norm) to the label vector $\mathbf{y} \in \mathbb{R}^m$ (4.6). The solution to this approximation problem is precisely the orthogonal projection of the vector $\mathbf{y}$ onto the subspace of $\mathbb{R}^m$ that is spanned by the columns of the feature matrix $\mathbf{X}$ (see Figure 4.3).

To solve the optimization problem (4.8), it is convenient to rewrite it as the quadratic problem

$$\min_{\mathbf{w} \in \mathbb{R}^n} \underbrace{(1/2)\mathbf{w}^T \mathbf{Q} \mathbf{w} - \mathbf{q}^T \mathbf{w}}_{=f(\mathbf{w})}$$
$$\text{with } \mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q} = (1/m)\mathbf{X}^T\mathbf{y}. \tag{4.9}$$

Since $f(\mathbf{w})$ is a differentiable and convex function, a necessary and sufficient condition for $\widehat{\mathbf{w}}$ to be a minimizer $f(\widehat{\mathbf{w}}) = \min_{\mathbf{w} \in \mathbb{R}^n} f(\mathbf{w})$ is the **zero-gradient condition** [14, Sec. 4.2.3]

$$\nabla f(\widehat{\mathbf{w}}) = \mathbf{0}. \tag{4.10}$$

Combining (4.9) with (4.10), yields the following necessary and sufficient condition for a

parameter vector $\widehat{\mathbf{w}}$ to solve the ERM (4.5),

$$(1/m)\mathbf{X}^T\mathbf{X}\widehat{\mathbf{w}} = (1/m)\mathbf{X}^T\mathbf{y}. \tag{4.11}$$

This condition can be rewritten as

$$(1/m)\mathbf{X}^T\big(\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}}\big) = \mathbf{0}. \tag{4.12}$$

As indicated in Figure 4.3, the optimality condition (4.12) requires the vector

$$\big(\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}}\big) = \big(\big(y^{(1)} - \hat{y}^{(1)}\big), \ldots, \big(y^{(m)} - \hat{y}^{(m)}\big)\big)^T,$$

whose entries are the prediction errors for the data points in the training set, to be orthogonal (or normal) to the subspace spanned by the columns of the feature matrix $\mathbf{X}$. In view of this geometric interpretation, we refer to (4.12) as a "normal equation".

It can be shown that, for any given feature matrix $\mathbf{X}$ and label vector $\mathbf{y}$, there always exists at least one optimal parameter vector $\widehat{\mathbf{w}}$ which solves (4.11). The optimal parameter vector might not be unique, i.e., there might be several different parameter vectors achieving the minimum in (4.5). However, every vector $\widehat{\mathbf{w}}$ which solves (4.11) achieves the same minimum empirical risk

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

Here, we used the orthogonal projection matrix $\mathbf{P} \in \mathbb{R}^{m\times m}$ on the linear span of the feature matrix $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})^T \in \mathbb{R}^{m\times n}$ (see (4.6)). The linear span of a matrix $\mathbf{A} = (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}) \in \mathbb{R}^{n\times m}$, denoted as $\mathrm{span}\big\{\mathbf{A}\big\}$, is the subspace of $\mathbb{R}^n$ consisting of all linear combinations of the columns $\mathbf{a}^{(r)} \in \mathbb{R}^n$ of $\mathbf{A}$.

If the columns of the feature matrix $\mathbf{X}$ (see (4.6)) are linearly independent, which implies that the matrix $\mathbf{X}^T\mathbf{X}$ is invertible, the projection matrix $\mathbf{P}$ is given explicitly as

$$\mathbf{P} = \mathbf{X}\big(\mathbf{X}^T\mathbf{X}\big)^{-1}\mathbf{X}^T.$$

Moreover, the solution of (4.11) is then unique and given by

$$\widehat{\mathbf{w}} = \big(\mathbf{X}^T\mathbf{X}\big)^{-1}\mathbf{X}^T\mathbf{y}. \tag{4.14}$$

The closed-form solution (4.14) requires the inversion of the $n \times n$ matrix $\mathbf{X}^T\mathbf{X}$.

Note that formula (4.14) is only valid if the matrix $\mathbf{X}^T\mathbf{X}$ is invertible. The feature matrix $\mathbf{X}$ is determined by the data points obtained in a ML application. Its properties are therefore not under the control of a ML method and it might well happen that the matrix $\mathbf{X}^T\mathbf{X}$ is not invertible. As a point in case, the matrix $\mathbf{X}^T\mathbf{X}$ cannot be invertible for any dataset containing fewer data points than the number of features used to characterize data points (this is referred to as high-dimensional data).

Moreover, the matrix $\mathbf{X}^T\mathbf{X}$ is not invertible if there are two co-linear features $x_j, x_{j'}$ such that $x_j = \beta x_{j'}$ holds for any data point with some constant $\alpha \in \mathbb{R}$. Indeed, the existence of two co-linear features implies that the (feature) matrix $\mathbf{X}$ has a non-trivial null-space which, in turn, implies that the matrix $\mathbf{X}^T\mathbf{X}$ cannot be invertible (see Exercise 4.9).

Let us now consider a dataset such that the feature matrix $\mathbf{X}$ is not full column-rank and, in turn, the matrix $\mathbf{X}^T\mathbf{X}$ is not invertible. In this case we cannot use (4.14) to compute the optimal parameter vector since the inverse of $\mathbf{X}^T\mathbf{X}$ does not exist. Moreover, in this case, there are infinitely many different weight vectors that solve (4.11). Each of these solutions result in a linear hypothesis with minimum average squared error loss on the training set. Which one of these optimal weight vectors should we pick ? Section 7.3 explains the benefits of using weight vectors with a small Euclidean norm. The optimal weight vector $\widehat{\mathbf{w}}$ solving the linear regression optimality condition (4.11) and having minimum Euclidean norm among all such vectors is given by

$$\widehat{\mathbf{w}} = \big(\mathbf{X}^T\mathbf{X}\big)^{\dagger}\mathbf{X}^T\mathbf{y}. \tag{4.15}$$

Here, $\big(\mathbf{X}^T\mathbf{X}\big)^{\dagger}$ denotes the pseudoinverse (or the Moore–Penrose inverse) of $\mathbf{X}^T\mathbf{X}$ (see [46, 45]).

Computing the (pseudo-)inverse of $\mathbf{X}^T\mathbf{X}$ can be computationally challenging for large number n of features. Figure 2.5 depicts a simple ML problem where the number of features is already in the millions. The computational complexity of inverting the matrix $\mathbf{X}^T\mathbf{X}$ depends crucially on its condition number. We refer to a matrix as ill-conditioned if its condition number is much larger than 1. In general, ML methods do not have any control on the condition number of the matrix $\mathbf{X}^T\mathbf{X}$. Indeed, this matrix is determined solely by the (features of the) data points fed into the ML method.

Section 5.4 will discuss a method for computing the optimal parameter vector $\widehat{\mathbf{w}}$ that does not require any matrix inversion. This method, referred to as GD constructs a sequence $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$ of increasingly accurate approximations of $\widehat{\mathbf{w}}$. This iterative method has two major benefits compared to evaluating the formula (4.14) using direct matrix inversion, such

as Gauss-Jordan elimination [46].

First, GD typically requires significantly fewer arithmetic operations compared to direct matrix inversion. This is crucial in modern ML applications involving large feature matrices. Second, GD does not break when the matrix $\mathbf{X}$ is not full rank and the formula (4.14) cannot be used any more.

4.4 ERM for Decision Trees

Consider ERM (4.3) for a regression problem with label space $\mathcal{Y} = \mathbb{R}$ and feature space $\mathcal{X} = \mathbb{R}^n$ and the hypothesis space defined by decision trees(see Section 3.10). In stark contrast to ERM for linear regression or logistic regression, ERM for decision trees amounts to a discrete optimization problem. Consider the particular hypothesis space $\mathcal{H}$ depicted in Figure 3.9. This hypothesis space contains a finite number of different hypothesis maps. Each individual hypothesis map corresponds to a particular decision tree.

For the small hypothesis space $\mathcal{H}$ in Figure 3.9, ERM is easy. Indeed, we just have to evaluate the empirical risk ("training error") $\widehat{L}(h)$ for each hypothesis in $\mathcal{H}$ and pick the one yielding the smallest empirical risk. However, when allowing for a very large (deep) decision tree, the computational complexity of exactly solving the ERM becomes intractable [65]. A popular approach to learn a decision tree is to use greedy algorithms which try to expand (grow) a given decision tree by adding new branches to leaf nodes in order to reduce the average loss on the training set (see [66, Chapter 8] for more details).

> The idea behind many decision tree learning methods is quite simple: try out expanding a decision tree by replacing a leaf node with a decision node (implementing another "test" on the feature vector) in order to reduce the overall empirical risk much as possible.

Consider the labeled dataset $\mathcal{D}$ depicted in Figure 4.4 and a given decision tree for predicting the label y based on the features $\mathbf{x}$. We might first try a hypothesis obtained from the simple tree shown in the top of Figure 4.4. This hypothesis does not allow to achieve a small average loss on the training set $\mathcal{D}$. Therefore, we might grow the tree by replacing a leaf node with a decision node. According to Figure 4.4, to so obtained larger decision tree provides a hypothesis that is able to perfectly predict the labels of the training set (it achieves zero empirical risk).

One important aspect of methods that learn a decision tree by sequentially growing the tree is the question of when to stop growing. A natural stopping criterion might be obtained

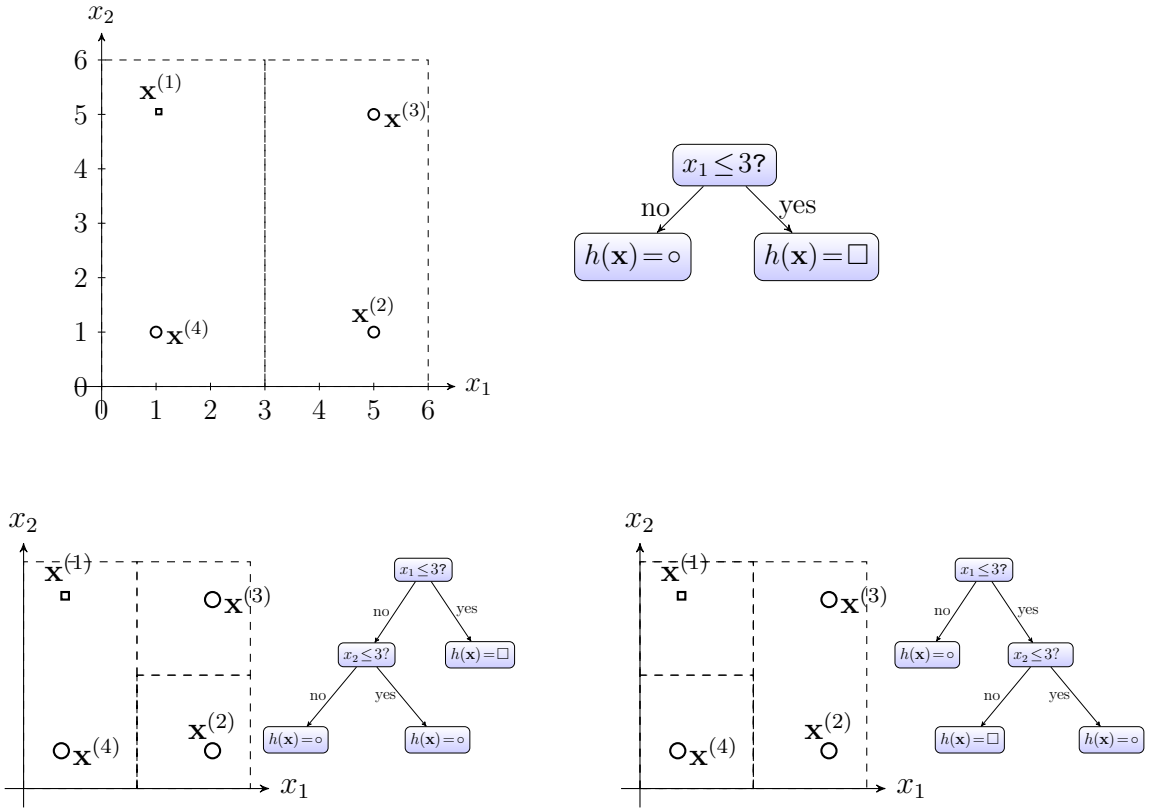


Figure 4.4: Consider a given labeled dataset and the decision tree in the top row. We then grow the decision tree by expanding one of its two leaf nodes. The bottom row shows the resulting decision trees, along with their decision boundaries. Each decision tree in the bottom row is obtained by expanding a different leaf node of the decision tree in the top row.

from the limitations in computational resources, i.e., we can only afford to use decision trees up to certain maximum depth. Besides the computational limitations, we also face statistical limitations for the maximum size of decision trees. ML methods that allow for very deep decision trees, which represent highly complicated maps, tend to overfit the training set (see Figure 3.10 and Chapter 7). Even if a deep decision tree incurs a small average loss on the training set, it might incur unacceptable high loss when predicting the labels of data points outside the training set.

4.5 ERM for Bayes Classifiers

The family of ML methods referred to as Bayes estimator uses the 0/1 loss (2.9) to measuring the quality of a classifier h. The resulting ERM is

$$\begin{aligned} \widehat{h} &= \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right) \\ &\overset{(2.9)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{i=1}^{m} \mathcal{I}(h(\mathbf{x}^{(i)}) \neq y^{(i)}). \end{aligned} \tag{4.16}$$

The objective function in this optimization problem is non-differentiable and non-convex (see Figure 4.2). This prevents us from using gradient-based methods (see Chapter 5) to solve (4.16).

We will now approach the ERM (4.16) via a different route by interpreting the data points $(\mathbf{x}^{(i)}, y^{(i)})$ as realizations of i.i.d. RVs with the common probability distribution $p(\mathbf{x}, y)$.

As discussed in Section 2.3, the empirical risk obtained using 0/1 loss approximates the error probability $p(\hat{y} \neq y)$ with the predicted label $\hat{y} = 1$ for $h(\mathbf{x}) > 0$ and $\hat{y} = -1$ otherwise (see (2.10)). Thus, we can approximate the ERM (4.16) as

$$\widehat{h} \overset{(2.10)}{\approx} \operatorname*{argmin}_{h \in \mathcal{H}} p(\hat{y} \neq y). \tag{4.17}$$

Note that the hypothesis h, which is the optimization variable in (4.17), enters into the objective function of (4.17) via the definition of the predicted label $\hat{y}$, which is $\hat{y} = 1$ if $h(\mathbf{x}) > 0$ and $\hat{y} = -1$ otherwise.

It turns out that if we would know the probability distribution $p(\mathbf{x}, y)$, which is required to compute $p(\hat{y} \neq y)$, the solution of (4.17) can be found via elementary Bayesian decision theory [115]. In particular, the optimal classifier $h(\mathbf{x})$ is such that $\hat{y}$ achieves the maximum

"a-posteriori" probability $p(\hat{y}|\mathbf{x})$ of the label being $\hat{y}$, given (or conditioned on) the features $\mathbf{x}$.

Since we typically do not know the probability distribution $p(\mathbf{x}, y)$, we have to estimate (or approximate) it from the observed data points $(\mathbf{x}^{(i)}, y^{(i)})$. This estimation is feasible if the data points can be considered (approximated) as realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{x}, y)$. We can then estimate (the parameters) of the probability distribution $p(\mathbf{x}, y)$ using maximum likelihood methods (see Section 3.12). For numeric features and labels, a widely-used parametric probability distribution $p(\mathbf{x}, y)$ is the multivariate normal (Gaussian) distribution. In particular, conditioned on the label y, the feature vector $\mathbf{x}$ is a Gaussian random vector with mean $\boldsymbol{\mu}_y$ and covariance $\boldsymbol{\Sigma}$,

$$p(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_y, \boldsymbol{\Sigma}).^{1} \tag{4.18}$$

The conditional expectation of the features $\mathbf{x}$, given (conditioned on) the label y of a data point, is $\boldsymbol{\mu}_1$ if $y = 1$, while for $y = -1$ the conditional mean of $\mathbf{x}$ is $\boldsymbol{\mu}_{-1}$. In contrast, the conditional covariance matrix $\boldsymbol{\Sigma} = \mathbb{E}\{(\mathbf{x} - \boldsymbol{\mu}_y)(\mathbf{x} - \boldsymbol{\mu}_y)^T|y\}$ of $\mathbf{x}$ is the same for both values of the label $y \in \{-1, 1\}$. The conditional probability distribution $p(\mathbf{x}|y)$ of the feature vector, given the label y, is multivariate normal. In contrast, the marginal distribution of the features $\mathbf{x}$ is a Gaussian mixture model (GMM). We will revisit GMMs later in Section 8.2 where we will see that they are a great tool for soft clustering.

For this probabilistic model of features and labels, the optimal classifier minimizing the error probability $p(\hat{y} \neq y)$ is $\hat{y} = 1$ for $h(\mathbf{x}) > 0$ and $\hat{y} = -1$ for $h(\mathbf{x}) \leq 0$ using the classifier map

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \text{ with } \mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}). \tag{4.19}$$

Carefully note that this expression is only valid if the matrix $\boldsymbol{\Sigma}$ is invertible.

We cannot implement the classifier (4.19) directly, since we do not know the true values of the class-specific mean vectors $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_{-1}$ and covariance matrix $\boldsymbol{\Sigma}$. Therefore, we have to replace those unknown parameters with some estimates $\hat{\boldsymbol{\mu}}_1$, $\hat{\boldsymbol{\mu}}_{-1}$ and $\widehat{\boldsymbol{\Sigma}}$. A principled

[1] We use the shorthand $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to denote the probability density function (pdf)

$$p(\mathbf{x}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\big(-(1/2)(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\big)$$

of a Gaussian random vector $\mathbf{x}$ with mean $\boldsymbol{\mu} = \mathbb{E}\{\mathbf{x}\}$ and covariance matrix $\boldsymbol{\Sigma} = \mathbb{E}\big\{(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T\big\}$.

approach is to use the maximum likelihood estimates (see (3.28))

$$\begin{aligned}
\hat{\boldsymbol{\mu}}_1 &= (1/m_1)\sum_{i=1}^{m} \mathcal{I}(y^{(i)}=1)\mathbf{x}^{(i)}, \\
\hat{\boldsymbol{\mu}}_{-1} &= (1/m_{-1})\sum_{i=1}^{m} \mathcal{I}(y^{(i)}=-1)\mathbf{x}^{(i)}, \\
\hat{\boldsymbol{\mu}} &= (1/m)\sum_{i=1}^{m} \mathbf{x}^{(i)}, \\
\text{and } \widehat{\boldsymbol{\Sigma}} &= (1/m)\sum_{i=1}^{m} (\mathbf{z}^{(i)}-\hat{\boldsymbol{\mu}})(\mathbf{z}^{(i)}-\hat{\boldsymbol{\mu}})^T, \qquad (4.20)
\end{aligned}$$

with $m_1 = \sum_{i=1}^{m} \mathcal{I}(y^{(i)} = 1)$ denoting the number of datapoints with label $y = 1$ (m_{-1} is defined similarly). Inserting the estimates (4.20) into (4.19) yields the implementable classifier

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \text{ with } \mathbf{w} = \widehat{\boldsymbol{\Sigma}}^{-1}(\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_{-1}). \qquad (4.21)$$

We highlight that the classifier (4.21) is only well-defined if the estimated covariance matrix $\widehat{\boldsymbol{\Sigma}}$ (4.20) is invertible. This requires to use a sufficiently large number of training datapoints such that $m \geq n$.

We derived the classifier (4.21) as an approximate solution to the ERM (4.16). The classifier (4.21) partitions the feature space $\mathbb{R}^n$ into two half-spaces. One half-space consists of feature vectors $\mathbf{x}$ for which the hypothesis (4.21) is non-negative and, in turn, $\hat{y} = 1$. The other half-space is constituted by feature vectors $\mathbf{x}$ for which the hypothesis (4.21) is negative and, in turn, $\hat{y} = -1$. Figure 2.9 illustrates these two half-spaces and the decision boundary between them.

The Bayes estimator (4.21) is another instance of a linear classifier like logistic regression and the SVM. Each of these methods learns a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, whose decision boundary (vectors $\mathbf{x}$ with $h(\mathbf{x}) = 0$) is a hyperplane (see Figure 2.9). However, these methods use different loss functions for assessing the quality of a particular linear hypothesis $h(\mathbf{x}) = \mathbf{wx}$ (which defined the decision boundary via $h(\mathbf{x}) = 0$). Therefore, these three methods typically learn classifiers with different decision boundaries.

For the estimator $\widehat{\boldsymbol{\Sigma}}$ (3.28) to be accurate (close to the unknown covariance matrix) we need a number of datapoints (sample size) which is at least of the order n^2. This sample size requirement might be infeasible for applications with only few datapoints available.

The maximum likelihood estimate $\widehat{\boldsymbol{\Sigma}}$ (4.20) is not invertible whenever $m < n$. In this case,

the expression (4.21) becomes useless. To cope with small sample size $m < n$ we can simplify the model (4.18) by requiring the covariance to be diagonal $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2)$. This is equivalent to modelling the individual features $x_1, \ldots, x_n$ of a data point as conditionally independent, given its label y. The resulting special case of a Bayes estimator is often referred to as a "naive Bayes" classifier.

We finally highlight that the classifier (4.21) is obtained using the generative model (4.18) for the data. Therefore, Bayes estimator belong to the family of generative ML methods which involve modelling the data generation. In contrast, logistic regression and the SVM do not require a generative model for the data points but aim directly at finding the relation between features $\mathbf{x}$ and label y of a data point. These methods belong therefore to the family of discriminative ML methods.

Generative methods such as those learning a Bayes estimator are preferable for applications with only very limited amounts of labeled data. Indeed, having a generative model such as (4.18) allows us to synthetically generate more labeled data by generating random features and labels according to the probability distribution (4.18). We refer to [105] for a more detailed comparison between generative and discriminative methods.

4.6 Training and Inference Periods

Some ML methods repeat the cycle in Figure 1 in a highly irregular fashion. Consider a large image collection which we use to learn a hypothesis about how cat images look like. It might be reasonable to adjust the hypothesis by fitting a model to the image collection. This fitting or training amounts to repeating the cycle in Figure 1 during some specific time period (the "training time") for a large number.

After the training period, we only apply the hypothesis to predict the labels of new images. This second phase is also known as inference period and might be much longer compared to the training period. Ideally, we would like to only have a very short training period to learn a good hypothesis and then only use the hypothesis for inference.

4.7 Online Learning

In it most basic form, ERM requires a given set of labeled data points, which we refer to as the training set. However, some ML methods can access data only in a sequential fashion. As a point in case, consider time series data such as daily minimum and maximum temperatures

recorded by a FMI weather station. Such a time series consists of a sequence of data points that are generated at successive time instants.

Online learning studies ML methods that learn (or optimize) a hypothesis incrementally as new data arrives. This mode of operation is quite different from ML methods that learn a hypothesis at once by solving an ERM problem. These different operation modes corresponds to different frequencies of iterating the basic ML cycle depicted in Figure 1. Online learning methods start a new cycle in Figure 1 whenever a new data point arrives (e.g., we have recorded the minimum and maximum temperate of a day that just ended).

We now present an online learning variant of linear regression (see Section 3.1) which is suitable for time series data with data points $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$ gathered sequentially (over time). In particular, the data points $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$ become available (are gathered) at time instants $t = 1, 2, 3 \ldots$.

Let us stack the feature vectors and labels of all data points available at time t into feature matrix $\mathbf{X}^{(t)}$ and label vector $\mathbf{y}^{(t)}$, respectively. The feature matrix and label vector for the first three time instants are

$$\begin{aligned} t = 1: \quad & \mathbf{X}^{(1)} := \big(\mathbf{x}^{(1)}\big)^T, & \mathbf{y}^{(1)} &= \big(y^{(1)}\big)^T, \\ t = 2: \quad & \mathbf{X}^{(2)} := \big(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}\big)^T, & \mathbf{y}^{(2)} &= \big(y^{(1)}, y^{(2)}\big)^T, \\ t = 3: \quad & \mathbf{X}^{(3)} := \big(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}\big)^T, & \mathbf{y}^{(3)} &= \big(y^{(1)}, y^{(2)}, y^{(3)}\big)^T. \end{aligned} \tag{4.22}$$

As detailed in Section 3.1, linear regression learns the weights $\mathbf{w}$ of a linear map $h(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ by minimizing the squared error loss $\big(y - h(\mathbf{x})\big)^2$. This informal goal of linear regression is made precise by the ERM problem (4.5) which defines the optimal weights via incurring minimum average squared error loss (empirical risk) on a given training set $\mathcal{D}$. These optimal weights are given by the solutions of (4.12). When the feature vectors of data points in $\mathcal{D}$ are linearly independent, we obtain the closed-form expression (4.14) for the optimal weights.

Inserting the feature matrix $\mathbf{X}^{(t)}$ and label vector $\mathbf{y}^{(t)}$ (4.22) into (4.14), yields

$$\widehat{\mathbf{w}}^{(t)} = \big(\big(\mathbf{X}^{(t)}\big)^T\mathbf{X}^{(t)}\big)^{-1}\big(\mathbf{X}^{(t)}\big)^T\mathbf{y}^{(t)}. \tag{4.23}$$

For each time instant we can evaluate the RHS of (4.23) to obtain the parameter vector $\widehat{\mathbf{w}}^{(t)}$ that minimizes the average squared error loss over all data points gathered up to time t. However, computing $\widehat{\mathbf{w}}^{(t)}$ via direct evaluation of the RHS in (4.23) for each new time instant t misses an opportunity for recycling computations done already at earlier time instants.

Let us now show how to (partially) reuse the computations used to evaluate (4.23) for

time t in the evaluation of (4.23) for the next time instant $t+1$. To this end, we first rewrite the matrix $\mathbf{Q}^{(t)} := \big(\mathbf{X}^{(t)}\big)^T \mathbf{X}^{(t)}$ as

$$\mathbf{Q}^{(t)} = \sum_{r=1}^{t} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T. \tag{4.24}$$

Since $\mathbf{Q}^{(t+1)} = \mathbf{Q}^{(t)} + \mathbf{x}^{(t+1)} \big(\mathbf{x}^{(t+1)}\big)^T$, we can use a well-known identity for matrix inverses (see [7, 94]) to obtain

$$\big(\mathbf{Q}^{(t+1)}\big)^{-1} = \big(\mathbf{Q}^{(t)}\big)^{-1} + \frac{\big(\mathbf{Q}^{(t)}\big)^{-1} \mathbf{x}^{(t+1)} \big(\mathbf{x}^{(t+1)}\big)^T \big(\mathbf{Q}^{(t)}\big)^{-1}}{1 - \big(\mathbf{x}^{(t+1)}\big)^T \big(\mathbf{Q}^{(t)}\big)^{-1} \mathbf{x}^{(t+1)}}. \tag{4.25}$$

Inserting (4.25) into (4.23) yields the following relation between optimal parameter vectors at consecutive time instants t and $t+1$,

$$\widehat{\mathbf{w}}^{(t+1)} = \widehat{\mathbf{w}}^{(t)} - \big(\mathbf{Q}^{(t+1)}\big)^{-1} \mathbf{x}^{(t+1)} \big(\big(\mathbf{x}^{(t+1)}\big)^T \widehat{\mathbf{w}}^{(t)} - y^{(t+1)}\big). \tag{4.26}$$

Note that neither evaluating the RHS of (4.26) nor evaluating the RHS of (4.25) requires to actually invert a matrix of with more than one entry (we can think of a scalar number as 1×1 matrix). In contrast, evaluating the RHS (4.23) requires to invert the matrix $\mathbf{Q}^{(t)} \in \mathbb{R}^{n \times n}$. We obtain an online algorithm for linear regression via computing the updates (4.26) and (4.25) for each new time instant t. Another online method for linear regression will be discussed at the end of Section 5.7.

4.8 Weighted ERM

Consider a ML method that uses some hypothesis space $\mathcal{H}$ and loss function L to measure the quality predictions obtained from a specific hypothesis when applied to a data point. A principled approach to learn a useful hypothesis is via ERM (4.3) using a training set

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

.

For some applications it might be useful to modify ERM (4.3) by weighting the data points. In particular, for each data point $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$ we specify a non-negative weight $q^{(i)} \in \mathbb{R}_+$. Weighted ERM is obtained from ERM (4.3) by replacing the average loss over the

training set with a weighted average loss,

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{m} q^{(i)} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right). \tag{4.27}$$

Note that we obtain ERM (4.3) as the special case of weighted ERM (4.27) for the weights $q^{(i)} = 1/m$.

We might interpret the sample weight $q^{(i)}$ as a measure for the importance or relevance of the data point $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ for the hypothesis $\hat{h}$ learnt via (4.27). The extreme case $q^{(i)} = 0$ means that the data point $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ becomes irrelevant for learning a hypothesis via (4.27). This could be useful if the data point $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ represents an outlier that violates the i.i.d. assumption which is satisfied by most of the other data points. Thus, using suitable weights in (4.27) could make the resulting ML method robust against outliers in the training set. Note that we have discussed another strategy (via the choice for the loss function) to achieve robustness against outliers in Section 3.3.

Another use-case of weighted ERM (4.27) is for applications where the risk of a hypothesis is defined using a probability distribution that is different form the probability distribution of the data points in the training set. Thus, the data points conform to an i.i.d. assumption with underlying probability distribution $p(\mathbf{x}, y)$. However, we would like to measure the quality of a hypothesis via the expected loss or risk using a different probability distribution $p'(\mathbf{x}, y)$,

$$\mathbb{E}_{p'}\left\{L\left((\mathbf{x}, y), h\right)\right\} = \int L\left((\mathbf{x}, y), h\right) dp'(\mathbf{x}, y). \tag{4.28}$$

Having a different probability distribution $p'(\mathbf{x}, y) (\neq p(\mathbf{x}, y)))$ to define the overall quality (risk) of a hypothesis might be beneficial for binary classification problems with imbalanced data. Indeed, using the average loss (which approximates the risk under $p(\mathbf{x}, y)$) might not be a useful quality measure if one class is over-represented in the training set (see Section 2.3.4). It can be shown that, under mild conditions, the weighted average loss in (4.27) approximates (4.28) when using the weights $q^{(i)} = p'\left(\mathbf{x}^{(i)}, y^{(i)}\right)/p\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ [12, Sec. 11.1.4].

4.9 Exercise

Exercise 4.1. Uniqueness in Linear Regression What conditions on a training set ensure that there is a unique optimal linear hypothesis map for linear regression?

Exercise 4.2. Uniqueness in Linear Regression II Linear regression uses the squared error loss (2.8) to measure the quality of a linear hypothesis map. We learn the weights $\mathbf{w}$ of a linear map via ERM using a training set $\mathcal{D}$ that consists of $m = 100$ data points. Each data point is characterized by $n = 5$ features and a numeric label. Is there a unique choice for the weights $\mathbf{w}$ that results in a linear predictor with minimum average squared error loss on the training set $\mathcal{D}$)?

Exercise 4.3. A Simple Linear Regression Problem. Consider a training set of m data points, each characterized by a single numeric feature x and numeric label y. We learn a hypothesis of the form $h(x) = x + b$ with some bias $b \in \mathbb{R}$. Can you write down a formula for the optimal b, that minimizes the average squared error loss on training data $\big(x^{(1)}, y^{(1)}\big), \ldots, \big(x^{(m)}, y^{(m)}\big)$.

Exercise 4.4. Simple Least Absolute Deviation Problem. Consider data points characterized by single numeric feature x and label y. We learn a hypothesis map of the form $h(x) = x + b$ with some bias $b \in \mathbb{R}$. Can you write down a formula for the optimal b, that minimizes the average absolute error on training data $\big(x^{(1)}, y^{(1)}\big), \ldots, \big(x^{(m)}, y^{(m)}\big)$.

Exercise 4.5. Polynomial Regression. Consider polynomial regression for data points with a single numeric feature $x \in \mathbb{R}$ and numeric label y. Here, polynomial regression is equivalent to linear regression using the transformed feature vectors $\mathbf{x} = \big(x^0, x^1, \ldots, x^{n-1}\big)^T$. Given a dataset $\mathcal{D} = \big(x^{(1)}, y^{(1)}\big), \ldots, \big(x^{(m)}, y^{(m)}\big)$, we construct the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big) \in \mathbb{R}^{m \times m}$ with its ith column given by the feature vector $\mathbf{x}^{(i)}$. Verify that this feature matrix is a Vandermonde matrix [44]? How is the determinant of the feature matrix related to the features and labels of data points in the dataset $\mathcal{D}$?

Exercise 4.6. Training Error is not Expected Loss. Consider a training set that consists of data points $\big(x^{(i)}, y^{(i)}\big)$, for $i = 1, \ldots, m = 100$, that are obtained as realizations of i.i.d. RVs. The common probability distribution of these RVs is defined by a random data point (x, y). The feature x of this random data point is a standard Gaussian RV with zero mean and unit variance. The label of a data point is modelled as $y = x + e$ with Gaussian noise $e \sim \mathcal{N}(0, 1)$. The feature x and noise e are statistically independent. We evaluate the

specific hypothesis $h(x) = 0$ (which outputs 0 no matter what the feature value x is) by the training error $E_t = (1/m) \sum_{i=1}^{m} \big(y^{(i)} - h\big(x^{(i)}\big)\big)^2$. Note that E_t is the average squared error loss (2.8) incurred by hypothesis h on the data points $\big(x^{(i)}, y^{(i)}\big)$, for $i = 1, \ldots, m = 100$. What is the probability that the training error E_t is at least 20 % larger than the expected (squared error) loss $\mathbb{E}\big\{\big(y - h(x)\big)^2\big\}$? What is the mean (expected value) and variance of the training error ?

Exercise 4.7. Optimization Methods as Filters. Let us consider a fictional (ideal) optimization method that can be represented as a filter $\mathcal{F}$. This filter $\mathcal{F}$ reads in a real-valued objective function $f(\cdot)$, defined for all parameter vectors $\mathbf{w} \in \mathbb{R}^n$. The output of the filter $\mathcal{F}$ is another real-valued function $\hat{f}(\mathbf{w})$ that is defined point-wise as

$$\hat{f}(\mathbf{w}) = \begin{cases} 1 & \text{, if } \mathbf{w} \text{ is a local minimum of } f(\cdot) \\ 0 & \text{, otherwise.} \end{cases} \tag{4.29}$$

Verify that the filter $\mathcal{F}$ is shift or translation invariant, i.e., $\mathcal{F}$ commutes with a translation $f'(\mathbf{w}) := f(\mathbf{w} + \mathbf{w}^{(o)})$ with an arbitrary but fixed (reference) vector $\mathbf{w}^{(o)} \in \mathbb{R}^n$.

Exercise 4.8. Linear Regression with Sample Weighting. Consider a linear regression method that uses ERM to learn weights $\widehat{\mathbf{w}}$ of a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The weights are learnt by minimizing the average squared error loss incurred by h on a training set that is constituted by the data points $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$ for $i = 1, \ldots, 100$.

Sometimes it is useful to assign sample-weights $q^{(i)}$ to the data points and learn $\widehat{\mathbf{w}}$. These sample-weights reflect varying levels of importance or relevance of different data points. For simplicity we use the sample weights $q^{(i)} = 2\alpha \in [0, 1]$ for $i = 1, \ldots, 50$ and $q^{(i)} = 2(1 - \alpha)$ for $i = 51, \ldots, 100$. Can you find a closed-form expression (similar to (4.14)) for the weights $\widehat{\mathbf{w}}^{(\alpha)}$ that minimize the weighted average squared error loss $f(\mathbf{w}) := (1/50) \sum_{i=1}^{50} \alpha\big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)^2 + (1/50) \sum_{i=51}^{100} (1 - \alpha)\big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)^2$ for different α?

Exercise 4.9. Co-Linear Features. Provide a rigorous proof of the fact that the $\mathbf{X}$ (see (4.6)) of a dataset cannot have full column rank if the data points have two co-linear features.

Chapter 5

Gradient-Based Learning

This chapter discusses an important family of optimization methods for solving ERM (4.4) with a parametrized hypothesis space (see Chapter 4.2). The common theme of these methods is to construct local approximations of the objective function in (4.4). These local approximations are obtained from the gradients of the objective function. Gradient-based methods have gained popularity recently as an efficient technique for tuning the parameters of deep nets within deep learning methods [48].

Section 5.1 discusses GD as the most basic form of gradient-based methods. The idea of GD is to update the weights by locally optimizing a linear approximation of the objective function. This update is referred to as a GD step and provides the main algorithmic primitive of gradient-based methods. One key challenge for a good use of gradient-based methods is the appropriate extend of the local approximations. This extent is controlled by a step size parameter that is used in the basic GD step. Section 5.2 discusses some approaches for choosing this step size. Section 5.3 discusses a second main challenge in using gradient-based methods which is to decide when to stop repeating the GD steps.

Section 5.4 and Section 5.5 spell out GD for two instances of ERM arising from linear regression and logistic regression, respectively. The beneficial effect of data normalization on the convergence speed of gradient-based methods is briefly discussed in Section 5.6. As explained in Section 5.7, the use of stochastic approximations enables gradient-based methods for applications involving massive amounts of data ("big data"). Section 5.8 develops intuition for advanced gradient-based methods that exploit the information gathered during previous iterations.

5.1 The Basic Gradient Step

Let us rewrite ERM (4.4) as the optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}) := (1/m)\sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right). \tag{5.1}$$

From now on we tacitly assume that each individual loss

$$f_i(\mathbf{w}) := L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right) \tag{5.2}$$

arising in (5.1) represents a differentiable function of the parameter vector $\mathbf{w}$. Trivially, differentiability of the components (5.2) implies differentiability of the overall objective function $f(\mathbf{w})$ (5.1).

Two important examples of ERM involving such differentiable loss functions are linear regression and logistic regression. In contrast, the hinge loss (2.11) used by the SVM results in a non-differentiable objective function $f(\mathbf{w})$ (5.1). However, it is possible to (significantly) extend the scope of gradient-based methods to non-differentiable functions by replacing the concept of a gradient with that of a subgradient.

Gradient based methods are iterative. They construct a sequence of parameter vectors $\mathbf{w}^{(0)} \rightarrow \mathbf{w}^{(1)} \ldots$ that hopefully converge to a minimizer $\overline{\mathbf{w}}$ of $f(\mathbf{w})$,

$$f(\overline{\mathbf{w}}) = \bar{f} := \min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}). \tag{5.3}$$

Note that there might be several different optimal parameter vectors $\overline{\mathbf{w}}$ that satisfy the optimality condition (5.3). We want the sequence generated by a gradient based method to converge towards any of them. The vectors $\mathbf{w}^{(r)}$ are (hopefully) increasingly, with increasing iteration r, more accurate approximation for a minimizer $\overline{\mathbf{w}}$ of (5.3).

Since the objective function $f(\mathbf{w})$ is differentiable, we can approximate it locally around the vector $\mathbf{w}^{(r)}$ using a tangent hyperplane that passes through the point $\left(\mathbf{w}^{(r)}, f\left(\mathbf{w}^{(r)}\right)\right) \in \mathbb{R}^{n+1}$. The normal vector of this hyperplane is given by $\mathbf{n} = (\nabla f\left(\mathbf{w}^{(r)}\right), -1)$ (see Figure 5.1). The first component of the normal vector is the gradient $\nabla f(\mathbf{w})$ of the objective function $f(\mathbf{w})$ evaluated at the vector $\mathbf{w}^{(r)}$. A main use of the gradient $\nabla f\left(\mathbf{w}^{(r)}\right)$ is to construct the linear approximation [119]

$$f(\mathbf{w}) \approx f\left(\mathbf{w}^{(r)}\right) + \left(\mathbf{w} - \mathbf{w}^{(r)}\right)^T \nabla f\left(\mathbf{w}^{(r)}\right) \text{ for } \mathbf{w} \text{ sufficiently close to } \mathbf{w}^{(r)}. \tag{5.4}$$

Requiring the objective function $f(\mathbf{w})$ in (5.4) to be differentiable is the same as requiring the validity of the local linear approximation (5.4) at every possible vector $\mathbf{w}^{(r)}$. It turns out that differentiability alone is not very helpful for the design and analysis of gradient based methods.

Gradient based methods are most useful for finding the minimum of differentiable functions $f(\mathbf{w})$ that are also smooth. Informally, a differentiable function $f(\mathbf{w})$ is smooth if the gradient $\nabla f(\mathbf{w})$ does not change too rapidly as a function of the argument $\mathbf{w}$. A quantitative version of the smoothness concept refers to a function as β-smooth if its gradient is Lipschitz continuous with Lipschitz constant $\beta > 0$ [16, Sec. 3.2],

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|. \tag{5.5}$$

Note that if a function $f(\mathbf{w})$ is β smooth, it is also β' smooth for any $\beta' > \beta$. The smallest β such that (5.5) is satisfied depends on the features and labels of data points used in (5.1) as well as on the choice for the loss function.

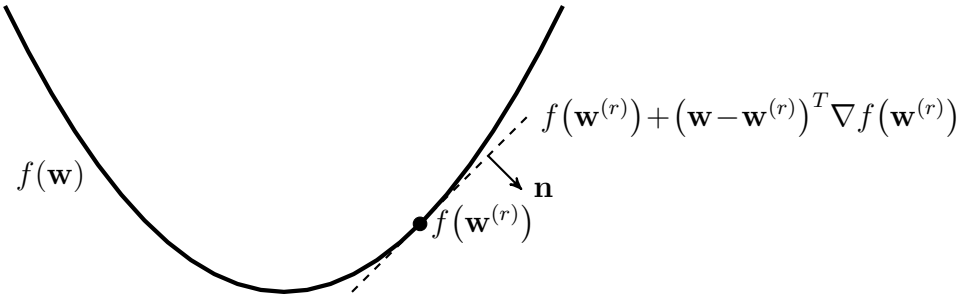


Figure 5.1: A differentiable function $f(\mathbf{w})$ can be approximated locally around a point $\mathbf{w}^{(r)}$ using a hyperplane whose normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(r)}\big), -1)$ is determined by the gradient $\nabla f\big(\mathbf{w}^{(r)}\big)$ [119].

Consider a current guess or approximation $\mathbf{w}^{(r)}$ for the optimal parameter vector $\overline{\mathbf{w}}$ (5.3). We would like to find a new (better) parameter vector $\mathbf{w}^{(r+1)}$ that has smaller objective value $f(\mathbf{w}^{(r+1)}) < f\big(\mathbf{w}^{(r)}\big)$ than the current guess $\mathbf{w}^{(r)}$. The approximation (5.4) suggests to choose the next guess $\mathbf{w} = \mathbf{w}^{(r+1)}$ such that $\big(\mathbf{w}^{(r+1)} - \mathbf{w}^{(r)}\big)^T \nabla f\big(\mathbf{w}^{(r)}\big)$ is negative. We can achieve this by the GD step

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha \nabla f(\mathbf{w}^{(r)}) \tag{5.6}$$

with a sufficiently small step size $\alpha > 0$. Figure 5.2 illustrates the GD step (5.6) which is the elementary computation of gradient based methods.

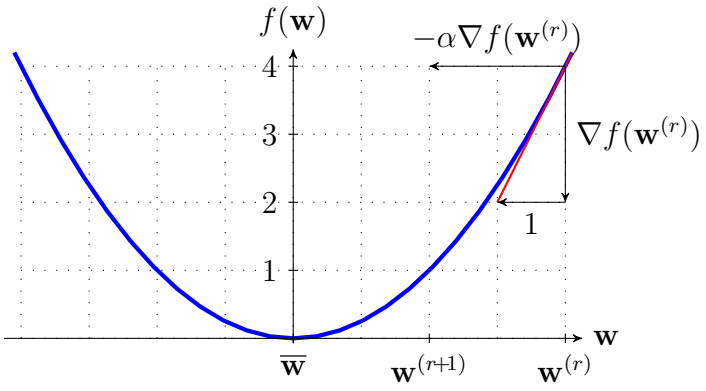


Figure 5.2: A GD step (5.6) updates a current guess or approximation $\mathbf{w}^{(r)}$ for the optimal parameter vector $\overline{\mathbf{w}}$ (5.3) by adding the correction term $-\alpha\nabla f(\mathbf{w}^{(r)})$. The updated parameter vector $\mathbf{w}^{(r+1)}$ is (typically) an improved approximation of the minimizer $\overline{\mathbf{w}}$.

The step size α in (5.6) must be sufficiently small to ensure the validity of the linear approximation (5.4). In the context of ML, the GD step size parameter α is also referred to as learning rate. Indeed, the step size α determines the amount of progress during a GD step towards learning the optimal parameter vector $\overline{\mathbf{w}}$.

We need to emphasize that the interpretation of the step size α as a learning rate is only useful when the step size is sufficiently small. Indeed, when increasing the step size α in (5.6) beyond a critical value (that depends on the properties of the objective function $f(\mathbf{w})$), the iterates (5.6) move away from the optimal parameter vector $\overline{\mathbf{w}}$. Nevertheless, from now on we will consequently use the term learning rate for α.

The idea of gradient-based methods is to repeat the GD step (5.6) for a sufficient number of iterations (repetitions) to obtain a sufficiently accurate approximation of the optimal parameter vector $\overline{\mathbf{w}}$ (5.3). It turns out that this is feasible for a sufficiently small learning rate and if the objective function is convex and smooth. Section 5.2 discusses precise conditions on the learning rate such that the iterates produced by the GD step converge to the optimum parameter vector, i.e., $\lim_{r\to\infty} f(\mathbf{w}^{(r)}) = f\big(\overline{\mathbf{w}}\big)$.

To implement the GD step (5.6) we need to choose a useful learning rate α. Moreover, executing the GD step (5.6) requires to compute the gradient $\nabla f(\mathbf{w}^{(r)})$. Both tasks can be computationally challenging as discussed in Section 5.2 and 5.7. For the objective function (5.1) obtained in linear regression and logistic regression, we can obtain closed-form expressions for the gradient $\nabla f(\mathbf{w})$ (see Sections 5.4 and 5.5).

In general, we do not have closed-form expressions for the gradient of the objective func-

tion (5.1) arising from a non-linear hypothesis space. One example for such a hypothesis space is obtained from a ANN, which is used by deep learning methods (see Section 3.11). The empirical success of deep learning methods might be partially attributed to the availability of an efficient algorithm for computing the gradient $\nabla f(\mathbf{w}^{(r)})$. This algorithm is known as back-propagation [48].

5.2 Choosing the Learning Rate

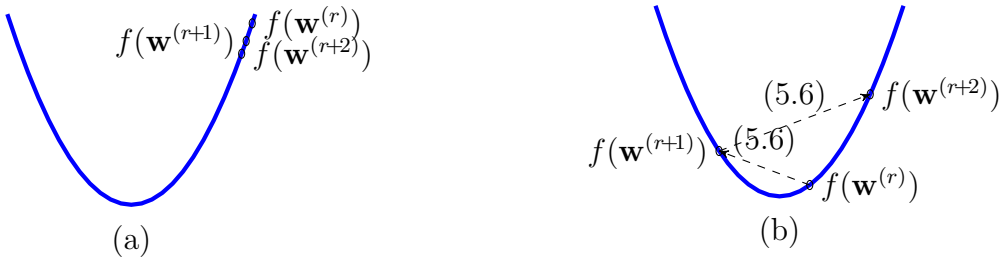


Figure 5.3: Effect of choosing bad values for the learning rate α in the GD step(5.6). (a) If the learning rate α in the GD step (5.6) is chosen too small, the iterations make very little progress towards the optimum or even fail to reach the optimum at all. (b) If the learning rate α is chosen too large, the iterates $\mathbf{w}^{(r)}$ might not converge at all (it might happen that $f(\mathbf{w}^{(r+1)}) > f(\mathbf{w}^{(r)})$!).

The choice of the learning rate α in the GD step (5.6) has a significant impact on the performance of the resulting ML methods. If we choose the learning rate α too large, the GD steps (5.6) diverge (see Figure 5.3-(b)) and, in turn, fail to converge towards the optimal weights $\overline{\mathbf{w}}$.

If we choose the learning rate α too small (see Figure 5.3-(a)), the updates (5.6) make only very little progress towards approximating the optimal parameter vector $\overline{\mathbf{w}}$. In applications that require real-time processing of data streams, it might be possible to repeat the GD steps only for a moderate number. If the learning rate is chosen too small, the updates (5.6) will fail to arrive at a good approximation of $\overline{\mathbf{w}}$ within an acceptable number of iterations.

Finding a (nearly) optimal choice for the learning rate α of GD can be a challenging task. Many sophisticated approaches for tuning the learning rate of gradient-based methods have been proposed [48, Chapter 8]. A detailed discussion of these approaches is beyond the scope of this book. We will instead discuss two sufficient conditions on the learning rate which

guarantee the convergence of the GD iterations to the optimum of a convex and smooth objective function (5.1).

The first condition applies to an objective function that is β-smooth (see (5.5)) with known constant β (not necessarily the smallest constant such that (5.5) holds). Then, the iterates $\mathbf{w}^{(r)}$ generated by the GD step (5.6) with a learning rate

$$\alpha < 2/\beta, \tag{5.7}$$

satisfy [102, Thm. 2.1.13]

$$f\big(\mathbf{w}^{(r)}\big) - \bar{f} \leq \frac{2(f\big(\mathbf{w}^{(0)}\big) - \bar{f}) \left\|\mathbf{w}^{(0)} - \overline{\mathbf{w}}\right\|_2^2}{2 \left\|\mathbf{w}^{(0)} - \overline{\mathbf{w}}\right\|_2^2 + r(f\big(\mathbf{w}^{(0)}\big) - \bar{f})\alpha(2 - \beta\alpha)}. \tag{5.8}$$

The bound (5.8) not only tells us that GD iterates converge to an optimal parameter vector but also characterize the convergence speed or rate. The sub-optimality $f\big(\mathbf{w}^{(r)}\big) - \min_{\mathbf{w}} f(\mathbf{w})$ in terms of objective function value decreases inversely (like "$1/r$") with the number r of GD steps (5.6). Convergence bounds like (5.8) can be used to specify a stopping criterion, i.e., to determine the number of GD steps to be computed (see Section 5.3).

The condition (5.7) and the bound (5.8) is only useful if we can verify β smoothness (5.5) assumption for a reasonable constant β. Verifying (5.8) only for a very large β results in the bound (5.8) being too loose (pessimistic). When we use a loose bound (5.8) to determine the number of GD steps, we might compute an unnecessary large number of GD steps (5.6).

One elegant approach to verify if a differentiable function $f(\mathbf{w})$ is β smooth (5.5) is via the Hessian matrix $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{n \times n}$ if it exists. The entries of this Hessian matrix are the second-order partial derivatives $\frac{\partial f(\mathbf{w})}{\partial w_j \partial w_{j'}}$ of the function $f(\mathbf{w})$.

Consider an objective function $f(\mathbf{w})$ (5.1) that is convex and twice-differentiable with psd Hessian $\nabla^2 f(\mathbf{w})$. If the maximum eigenvalue $\lambda_{\max}\big(\nabla^2 f(\mathbf{w})\big)$ of the Hessian is upper bounded uniformly (for all $\mathbf{w}$) by the constant $\beta > 0$, then $f(\mathbf{w})$ is β smooth (5.5) [16]. This implies, in turn via (5.7), the sufficient condition

$$\alpha < \frac{2}{\lambda_{\max}\big(\nabla^2 f(\mathbf{w})\big)} \text{ for all } \mathbf{w} \in \mathbb{R}^n \tag{5.9}$$

for the GD learning rate such that the GD steps converge to the minimum of the objective function $f(\mathbf{w})$.

It is important to note that the condition (5.9) guarantees convergence of the GD steps

for any possible initialization $\mathbf{w}^{(0)}$. Note that the usefulness of the condition (5.9) depends on the difficulty of computing the Hessian matrix $\nabla^2 f(\mathbf{w})$. Section 5.4 and Section 5.5 will present closed-form expressions for the Hessian of the objective function (5.1) obtained for linear regression and logistic regression. These closed-form expressions involve the feature vectors and labels of the data points in the training set $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$ used in (5.1).

While it might be computationally challenging to determine the maximum (in absolute value) eigenvalue $\lambda_{\max}\big(\nabla^2 f(\mathbf{w})\big)$ for arbitrary $\mathbf{w}$, it might still be feasible to find an upper bound U for it. If we know such an upper bound $U \geq \lambda_{\max}\big(\nabla^2 f(\mathbf{w})\big)$ (valid for all $\mathbf{w} \in \mathbb{R}^n$), the learning rate $\alpha = 1/U$ still ensures convergence of the GD steps (5.6).

Up to know we have assumed a fixed (constant) learning rate α that is used for each repetition of the GD steps (5.6). However, it might be useful to vary or adjust the learning rate as the GD steps (5.6) proceed. Thus, we might use a different learning rate α_r for each iteration r of (5.6). Such a varying learning rate is useful for a variant of GD that uses stochastic approximation (see Section 5.7). However, we might use a varying learning rate also to avoid the burden of verifying β smoothness (5.5) with a tight (small) β. The GD steps (5.6) with the learning rate $\alpha_r := 1/r$ converge to the optimal parameter vector $\overline{\mathbf{w}}$ as long as we can ensure a bounded gradient $\|\nabla f(\mathbf{w})\| \leq U$ for a sufficiently large neighbourhood of $\overline{\mathbf{w}}$ [102].

5.3 When To Stop?

One main challenge in the successful application of GD is to decide when to stop iterating (or repeating) the GD step (5.6). Maybe the most simple approach is to monitor the decrease in the objective function $f(\mathbf{w}^{(r)})$ and to stop if the decrease $f(\mathbf{w}^{(r-1)}) - f(\mathbf{w}^{(r)})$ falls below a threshold. However, the ultimate goal of a ML method is not to minimize the objective function $f(\mathbf{w})$ in (5.1). Indeed, the objective function represents the average loss of a hypothesis $h^{(\mathbf{w})}$ incurred on a training set. However, the ultimate goal of a ML method is to learn a parameter vector $\mathbf{w}$ such that the resulting hypothesis accurately predicts any data point, including those outside the training set.

We will see in Chapter 6 how to use validation techniques to probe a hypothesis outside the training set. These validation techniques provide a validation error $\tilde{f}(\mathbf{w})$ that estimates the average loss of a hypothesis with parameter vector $\mathbf{w}$. Early stopping techniques monitor the validation error $\tilde{f}(\mathbf{w}^{(r)})$ as the GD iterations r proceed to decide when to stop iterating.

Another possible stopping criterion is to use a fixed number of iterations or GD steps. This fixed number of iterations can be chosen based on convergence bounds such as (5.8) in order to guarantee a prescribed sub-optimality of the final iterate $\mathbf{w}^{(r)}$. A slightly more convenient convergence bound can be obtained from (5.8) when using the the learning rate $\alpha = 1/\beta$ in the GD step (5.6) [16],

$$f\big(\mathbf{w}^{(r)}\big) - \bar{f} \leq \frac{2\beta \left\|\mathbf{w}^{(0)} - \overline{\mathbf{w}}\right\|_2^2}{r} \text{ for } r = 1, 2, \ldots. \tag{5.10}$$

5.4 GD for Linear Regression

We now present a GD method for linear regression (see Section 3.1). This method learns the parameter vector for a linear hypothesis

$$h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}. \tag{5.11}$$

The ERM principle from Section 4.3 tells us to choose the parameter vector $\mathbf{w}$ in (5.11) by minimizing the average squared error loss (2.8)

$$\widehat{L}(h^{(\mathbf{w})}|\mathcal{D}) \stackrel{(4.4)}{=} (1/m) \sum_{i=1}^{m} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2. \tag{5.12}$$

The average squared error loss (5.12) is computed by applying the predictor $h^{(\mathbf{w})}(\mathbf{x})$ to labeled data points in a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$. An optimal parameter vector $\overline{\mathbf{w}}$ for (5.11) is obtained as

$$\overline{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n} f(\mathbf{w}) \text{ with } f(\mathbf{w}) = (1/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\big)^2. \tag{5.13}$$

The objective function $f(\mathbf{w})$ in (5.13) is convex and smooth. We can therefore use GD (5.6) to solve (5.13) iteratively, i.e., by constructing a sequence of parameter vectors that converge to an optimal parameter vector $\overline{\mathbf{w}}$. To implement GD, we need to compute the gradient $\nabla f(\mathbf{w})$.

The gradient of the objective function in (5.13) is given by

$$\nabla f(\mathbf{w}) = -(2/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\big) \mathbf{x}^{(i)}. \tag{5.14}$$

By inserting (5.14) into the basic GD iteration (5.6), we obtain Algorithm 1.

Algorithm 1 Linear regression via GD

Input: dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$; 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: $\mathbf{w}^{(r)} := \mathbf{w}^{(r-1)} + \alpha(2/m) \sum_{i=1}^{m} \big(y^{(i)} - \big(\mathbf{w}^{(r-1)}\big)^T \mathbf{x}^{(i)}\big)\mathbf{x}^{(i)}$ (do a GD step (5.6))
4: **until** stopping criterion met
Output: $\mathbf{w}^{(r)}$ (which approximates $\overline{\mathbf{w}}$ in (5.13))

Let us have a closer look on the update in step 3 of Algorithm 1, which is

$$\mathbf{w}^{(r)} := \mathbf{w}^{(r-1)} + \alpha(2/m) \sum_{i=1}^{m} \big(y^{(i)} - \big(\mathbf{w}^{(r-1)})^T \mathbf{x}^{(i)}\big)\mathbf{x}^{(i)}. \tag{5.15}$$

The update (5.15) has an appealing form as it amounts to correcting the previous guess (or approximation) $\mathbf{w}^{(r-1)}$ for the optimal parameter vector $\overline{\mathbf{w}}$ by the correction term

$$(2\alpha/m) \sum_{i=1}^{m} \underbrace{\big(y^{(i)} - \big(\mathbf{w}^{(r-1)})^T \mathbf{x}^{(i)}\big)}_{e^{(i)}} \mathbf{x}^{(i)}. \tag{5.16}$$

The correction term (5.16) is a weighted average of the feature vectors $\mathbf{x}^{(i)}$ using weights $(2\alpha/m) \cdot e^{(i)}$. These weights consist of the global factor $(2\alpha/m)$ (that applies equally to all feature vectors $\mathbf{x}^{(i)}$) and a sample-specific factor $e^{(i)} = (y^{(i)} - \big(\mathbf{w}^{(r-1)})^T \mathbf{x}^{(i)})$, which is the prediction (approximation) error obtained by the linear predictor $h^{(\mathbf{w}^{(r-1)})}(\mathbf{x}^{(i)}) = \big(\mathbf{w}^{(r-1)})^T \mathbf{x}^{(i)}$ when predicting the label $y^{(i)}$ from the features $\mathbf{x}^{(i)}$.

We can interpret the GD step (5.15) as an instance of "learning by trial and error". Indeed, the GD step amounts to first "trying out" (trial) the predictor $h(\mathbf{x}^{(i)}) = \big(\mathbf{w}^{(r-1)})^T \mathbf{x}^{(i)}$. The predicted values are then used to improve the weight vector $\mathbf{w}^{(r-1)}$ based on the error $e^{(i)} = y^{(i)} - \big(\mathbf{w}^{(r-1)})^T \mathbf{x}^{(i)}$.

The choice of the learning rate α used for Algorithm 1 can be based on the condition (5.9) with the Hessian $\nabla^2 f(\mathbf{w})$ of the objective function $f(\mathbf{w})$ underlying linear regression (see (5.13)). This Hessian is given explicitly as

$$\nabla^2 f(\mathbf{w}) = (1/m)\mathbf{X}^T \mathbf{X}, \tag{5.17}$$

with the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T \in \mathbb{R}^{m \times n}$ (see (4.6)). Note that the Hessian (5.17) does not depend on the parameter vector $\mathbf{w}$.

Comparing (5.17) with (5.9), one particular strategy for choosing the learning rate in Algorithm 1 is to (i) compute the matrix product $\mathbf{X}^T\mathbf{X}$, (ii) compute the maximum eigenvalue $\lambda_{\max}\big((1/m)\mathbf{X}^T\mathbf{X}\big)$ of this product and (iii) set the learning rate to $\alpha = 1/\lambda_{\max}\big((1/m)\mathbf{X}^T\mathbf{X}\big)$.

While it might be computationally difficult to compute the maximum eigenvalue of the psd matrix $(1/m)\mathbf{X}^T\mathbf{X}$ exactly, we might be able to find an upper bound U such that[1]

$$\lambda_{\max}\big((1/m)\mathbf{X}^T\mathbf{X}\big) \leq U. \tag{5.18}$$

In some applications, we might have prior knowledge about the data points and their features which allows us to find a reasonable upper bound U with little computational effort. Given an upper bound U in (5.18), we can ensure convergence of the GD steps when using learning rate $\alpha = 1/U$.

Consider a dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$ with normalized features, i.e., $\|\mathbf{x}^{(i)}\| = 1$ for all $i = 1, \ldots, m$. It can then be shown that $\lambda_{\max}\big((1/m)\mathbf{X}^T\mathbf{X}\big) \leq 1$, i.e., we can use the upper bound $U = 1$ in (5.18). Thus, in this case, we can ensure convergence of the iterates $\mathbf{w}^{(r)}$ (see (5.15)) by choosing the learning rate $\alpha = 1$.

Time-Data Tradeoffs. The number of GD steps required by Algorithm 1 to ensure a prescribed sub-optimality depends crucially on the condition number of $\mathbf{X}^T\mathbf{X}$. What can we say about the condition number? In general, we have not control over this quantity as the matrix $\mathbf{X}$ consists of the feature vectors of arbitrary data points. However, it is often useful to model the feature vectors as realizations of i.i.d. random vectors. It is then possible to bound the probability of the feature matrix having a sufficiently small condition number. These bounds can then be used to choose the step-size such that convergence is guaranteed with sufficiently large probability. The usefulness of these bounds typically depends on the ratio n/m. For increasing sample-size, these bounds allow to use larger step-sizes and, in turn, result in faster convergence of GD algorithm. Thus, we obtain a trade-off between the runtime of Algorithm 1 and the number of data points that we feed into it [107].

[1]The problem of computing a full eigenvalue decomposition of $\mathbf{X}^T\mathbf{X}$ has essentially the same complexity as ERM via directly solving (4.11), which we want to avoid by using the "cheaper" GD Algorithm 1.

5.5 GD for Logistic Regression

Logistic regression learns a linear hypothesis $h^{(\mathbf{w})}$ that is used to classify data points by predicting their binary label. The quality of such a linear classifier is measured by the logistic loss (2.12). The ERM principle suggest to learn the parameter vector $\mathbf{w}$ by minimizing the average logistic loss (3.16) obtained for a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$. The training set consists of data points with features $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and binary labels $y^{(i)} \in \{-1, 1\}$.

We can rewrite ERM for logistic regression as the optimization problem

$$\overline{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n} f(\mathbf{w})$$
$$\text{with } f(\mathbf{w}) = (1/m) \sum_{i=1}^{m} \log \big(1+\exp\big(-y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)}\big)\big). \tag{5.19}$$

The objective function $f(\mathbf{w})$ is differentiable and therefore we can use GD (5.6) to solve (5.19). We can write down the gradient of the objective function in (5.19) in closed-form as

$$\nabla f(\mathbf{w}) = (1/m) \sum_{i=1}^{m} \frac{-y^{(i)}}{1+\exp(y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)})} \mathbf{x}^{(i)}. \tag{5.20}$$

Inserting (5.20) into the GD step (5.6) yields Algorithm 2.

Algorithm 2 Logistic regression via GD

Input: labeled dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ containing feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and labels $y^{(i)} \in \mathbb{R}$; 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: $\mathbf{w}^{(r)} := \mathbf{w}^{(r-1)} + \alpha(1/m) \sum_{i=1}^{m} \frac{y^{(i)}}{1+\exp\left(y^{(i)} \left(\mathbf{w}^{(r-1)}\right)^T \mathbf{x}^{(i)}\right)} \mathbf{x}^{(i)}$ (do a GD step (5.6))

4: **until** stopping criterion met

Output: $\mathbf{w}^{(r)}$, which approximates a solution $\overline{\mathbf{w}}$ of (5.19))

Let us have a closer look on the update in step (3) of Algorithm 2. This step amounts to computing

$$\mathbf{w}^{(r)} := \mathbf{w}^{(r-1)} + \alpha(1/m) \sum_{i=1}^{m} \frac{y^{(i)}}{1+\exp\left(y^{(i)} \left(\mathbf{w}^{(r-1)}\right)^T \mathbf{x}^{(i)}\right)} \mathbf{x}^{(i)}. \tag{5.21}$$

Similar to the GD step (5.15) for linear regression, also the GD step (5.21) for logistic regression can be interpreted as an implementation of the trial-and-error principle. Indeed, (5.21) corrects the previous guess (or approximation) $\mathbf{w}^{(r-1)}$ for the optimal parameter vector $\overline{\mathbf{w}}$ by the correction term

$$(\alpha/m)\sum_{i=1}^{m} \underbrace{\frac{y^{(i)}}{1+\exp(y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)})}}_{e^{(i)}}\mathbf{x}^{(i)}. \tag{5.22}$$

The correction term (5.22) is a weighted average of the feature vectors $\mathbf{x}^{(i)}$. The feature vector $\mathbf{x}^{(i)}$ is weighted by the factor $(\alpha/m)\cdot e^{(i)}$. These weighting factors are a product of the global factor (α/m) that applies equally to all feature vectors $\mathbf{x}^{(i)}$. The global factor is multiplied by a data point-specific factor $e^{(i)} = \frac{y^{(i)}}{1+\exp(y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)})}$, which quantifies the error of the classifier $h^{(\mathbf{w}^{(r-1)})}(\mathbf{x}^{(i)}) = \big(\mathbf{w}^{(r-1)}\big)^T\mathbf{x}^{(i)}$ for a single data point with true label $y^{(i)} \in \{-1,1\}$ and features $\mathbf{x}^{(i)} \in \mathbb{R}^n$.

We can use the sufficient condition (5.9) for the convergence of GD steps to guide the choice of the learning rate α in Algorithm 2. To apply condition (5.9), we need to determine the Hessian $\nabla^2 f(\mathbf{w})$ matrix of the objective function $f(\mathbf{w})$ underlying logistic regression (see (5.19)). Some basic calculus reveals (see [58, Ch. 4.4.])

$$\nabla^2 f(\mathbf{w}) = (1/m)\mathbf{X}^T\mathbf{D}\mathbf{X}. \tag{5.23}$$

Here, we used the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}\big)^T \in \mathbb{R}^{m\times n}$ (see (4.6)) and the diagonal matrix $\mathbf{D} = \mathrm{diag}\{d_1,\ldots,d_m\} \in \mathbb{R}^{m\times m}$ with diagonal elements

$$d_i = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}^{(i)})}\left(1-\frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}^{(i)})}\right). \tag{5.24}$$

We highlight that, in contrast to the Hessian (5.17) of the objective function arising in linear regression, the Hessian (5.23) of logistic regression varies with the parameter vector $\mathbf{w}$. This makes the analysis of Algorithm 2 and the optimal choice for the learning rate α more difficult compared to Algorithm 1. At least, we can ensure convergence of (5.21) (towards a solution of (5.19)) for the learning rate $\alpha = 1$ if we normalize feature vectors such that $\|\mathbf{x}^{(i)}\| = 1$. This follows from the fact the diagonal entries (5.24) take values in the interval $[0,1]$.

5.6 Data Normalization

The number of GD steps (5.6) required to reach the minimum (within a prescribed accuracy) of the objective function (4.5) depends crucially on the condition number [16, 68]

$$\kappa(\mathbf{X}^T\mathbf{X}) := \lambda_{\max}/\lambda_{\min}. \tag{5.25}$$

Here, we use the largest and smallest eigenvalue of the matrix $\mathbf{X}^T\mathbf{X}$, denoted as $\lambda_{\max}$ and $\lambda_{\min}$, respectively. The condition number (5.25) is only well-defined if the columns of the feature matrix $\mathbf{X}$ (4.6) (which are the feature vectors $\mathbf{x}^{(i)}$), are linearly independent. In this case the condition number is lower bounded as $1 \leq \kappa(\mathbf{X}^T\mathbf{X})$.

It can be shown that the GD steps (5.6) converge faster for smaller condition number $\kappa(\mathbf{X}^T\mathbf{X})$ [68]. Thus, GD will be faster for datasets with a feature matrix $\mathbf{X}$ such that $\kappa(\mathbf{X}^T\mathbf{X}) \approx 1$. It is therefore often beneficial to pre-process the feature vectors using a normalization (or standardization) procedure as detailed in Algorithm 3.

Algorithm 3 "Data Normalization"

Input: labeled dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$

1: remove sample means $\widehat{\mathbf{x}} = (1/m)\sum_{i=1}^m \mathbf{x}^{(i)}$ from features, i.e.,

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

2: normalise features to have unit variance,

$$\hat{x}_j^{(i)} := x_j^{(i)}/\hat{\sigma} \text{ for } j = 1, \ldots, n \text{ and } i = 1, \ldots, m$$

with the empirical (sample) variance $\hat{\sigma}_j^2 = (1/m)\sum_{i=1}^m \left(x_j^{(i)}\right)^2$

Output: normalized feature vectors $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^m$

Algorithm 3 transforms the original feature vectors $\mathbf{x}^{(i)}$ into new feature vectors $\widehat{\mathbf{x}}^{(i)}$ such that the new feature matrix $\widehat{\mathbf{X}} = (\widehat{\mathbf{x}}^{(1)}, \ldots, \widehat{\mathbf{x}}^{(m)})^T$ is better conditioned than the original feature matrix, i.e., $\kappa(\widehat{\mathbf{X}}^T\widehat{\mathbf{X}}) < \kappa(\mathbf{X}^T\mathbf{X})$.