# Chapter 2
# Components of ML


This book portrays ML as combinations of three components (see Fig. 2.1):

- **data** as collections of individual data points that are characterized by features (see Sect. 2.1.1) and labels (see Sect. 2.1.2)
- a model or hypothesis space that consists of computationally feasible hypothesis maps from feature space to label space (see Sect. 2.2)
- a loss function (see Sect. 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 the three above components.

We detail in Chap. 3 how some of the most popular ML methods, including linear regression (see Sect. 3.1) as well as deep learning methods (see Sect. 3.11), are obtained by specific design choices for the three components. This chapter discusses in some depth the role of and the individual components of ML and their combination in ML 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, individual trees within a forest,



A. Jung, *Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-16-8193-6_2

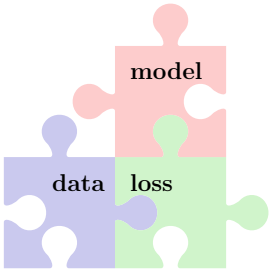



**Fig. 2.1** ML methods fit a model to data via minimizing a loss function

**Fig. 2.2** Snapshot taken at the beginning of a mountain hike

individual forests within a collection of forests. Mountain hikers might be interested in datapoints that represent different hiking tours (see Fig. 2.2).

We use the concept of datapoints in a very abstract and therefore highly flexible manner. Datapoints 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 datapoints as images. When developing a recommendation system we might define datapoints to represent customers. In the development of new drugs we might use data points to represent different diseases. The view in this book is that the meaning of definition of datapoints should be considered as 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) datapoints. These estimates become more accurate for an increasing number of datapoints used for computing the average. A key parameter of a dataset is the number $m$ of individual datapoints it contains. The number of datapoints within a dataset is also referred to as the sample size. Statistically, 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 datapoint that represents a vaccine. A full characterization of such a datapoint 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. Depending on the application domain, we might refer to labels also as a **target** or the **output variable**. The features of a data point are sometimes also referred to as **input variables**.

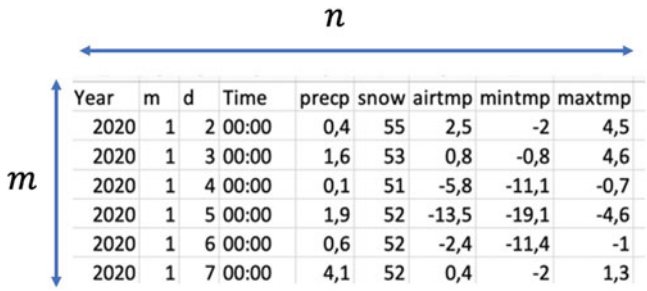
The distinction between features and labels is somewhat blurry. The same property of a data point might be used as a feature in one application, while it might be used as a label in another application. As an example, consider feature learning for datapoints representing images. One approach to learn representative features of an image is to use some of the image pixels as the label or target pixels. We can then learn new features by learning a feature map that allows us to predict the target pixels.

To further illustrate the blurry distinction between features and labels, consider 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 datapoints. However, few of these properties are unknown (missing) for a small set of datapoints (e.g., due to broken sensors). We could then define the properties which are missing for some datapoints 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].

Figure 2.3 illustrates two key parameters of a dataset. The first parameter is the sample size $m$, i.e., the number of individual data points that constitute the dataset. The second key parameter 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 Chap. 6.

### *2.1.1 Features*

Similar to the definition of datapoints, also the choice of which properties to be used as their features is a design choice. In general, features are properties of a datapoint that can be computed or measured easily. However, this is a highly informal characterization since there no universal criterion for the difficulty of computing of measuring a property of datapoints. The task of choosing which properties to use




| 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 |

**Fig. 2.3** Two main parameters 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 datapoints. The behaviour of ML methods typically depends crucially on the ratio $m/n$

as features of data points might be the most challenging part in the 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 Fig. 2.4).

The feature construction for images depicted in Fig. 2.4 can be extended to other types of data points as long as they can be visualized efficiently. As a case in point, we might visualize an audio recording using an intensity plot of its spectrogram (see Fig. 2.5). We can then use the pixel RGB intensities of this intensity plot 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 image segmentation to cluster the dataset (see Chap. 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

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

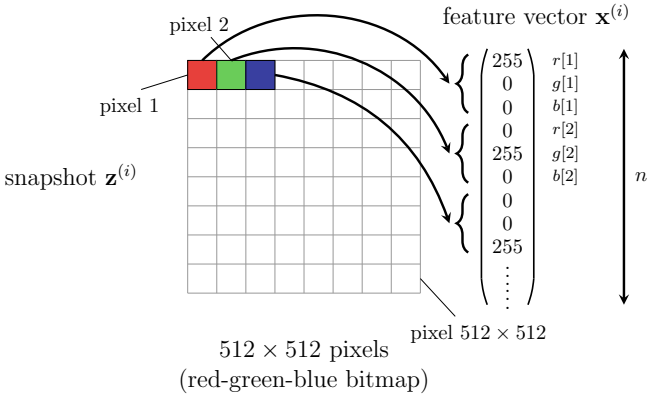



**Fig. 2.4** If the snapshot $\mathbf{z}^{(i)}$ is stored as a $512 \times 512$ RGB bitmap, we could use as features $\mathbf{x}^{(i)} \in \mathbb{R}^n$ the red-, green- and blue component of each pixel in the snapshot. The length of the feature vector would then be $n = 3 \cdot 512 \cdot 512 \approx 786000$

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 Fig. 2.4).

We refer to the set of possible feature vectors of datapoints arising in some ML application 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 datapoint 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 [2]. 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. Section 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 datapoints in many application domains. Consider a datapoint 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 datapoint. This results in more than $10^9$ features (per year!). If we stack all those features into a feature vector $\mathbf{x}$, its length $n$ would be of the order of $10^9$.

As indicated above, many important ML applications involve datapoints represented by very long feature vectors. To process such high-dimensional data, modern

ML methods rely on concepts from high-dimensional statistics [3, 4]. 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 [5].

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 Chap. 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 datapoints. As a rule of thumb, the number $m$ of (labeled) datapoints used to train a ML method should be much larger than the number $n$ of numeric features (see Fig. 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) datapoints 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 datapoints representing an image. 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 datapoints.

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 Chap. 8 to decompose a dataset into few groups or clusters that consist of similar data points.

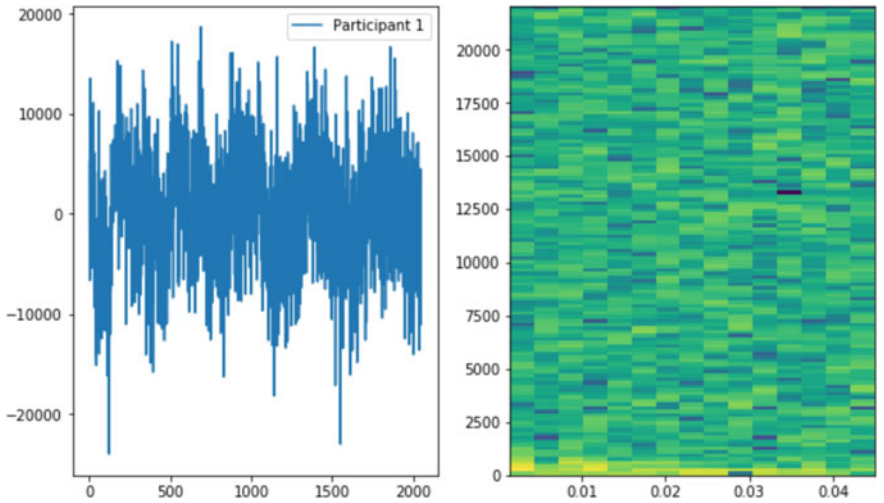



**Fig. 2.5** Two visualizations of an audio recording obtained from a line plot of the signal amplitudes and by the spectrogram of the audio recording

Throughout this book we will mainly use the feature space $\mathbb{R}^n$ with dimension $n$ being the number of features of a datapoint. 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 the data generated 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.

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].

Audio recordings are typically available a sequence of signal amplitudes $a_t$ collected regularly at time instants $t = 1, \ldots, n$ with sampling frequency $\approx 44$ 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 2.5 depicts two possible visualizations of an audio signal obtained from a line plot of the signal amplitudes (as a function of time index $t$) or an intensity plot of the spectrogram [7, 8].

### 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.

As already mentioned, the distinction of data point properties into labels and features is blurry. Roughly speaking, labels are properties of datapoints that might only be determined with the help of human experts. For datapoints 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 datapoint that represents some hike, at the start of which the snapshot in Fig. 2.2 has been taken. The features of this datapoint could be the red, green and blue (RGB) intensities of each pixel in the snapshot in Fig. 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 datapoint (which represents 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 visible 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 maleficent, 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 datapoint 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 $j$th 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 Fig. 2.4). 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 area 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 thresholding, $\hat{y} := 1$ if $\hat{y}' \geq 0$ whereas $\hat{y} := -1$ otherwise. A prominent example for this link between regression and classification is logistic regression which is discussed in Sect. 3.6. Despite its name, logistic regression is a binary classification method.

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 handling a water thermometer at certain locations in a lake. In other applications, acquiring labels might require sending out a team of marine biologists to the Baltic sea [9], running a particle physics experiment at the European organization for nuclear research (CERN) [10], running animal testing in pharmacology [11].

Let us also point out online market places for human labelling workforce [12]. 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 datapoints. This labeling work might amount to marking images that show a cat.

Many applications involve datapoints whose features can be determined easily, but whose labels are known for few datapoints 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 [13]. ML methods for speech recognition and machine translation can make use of massive labeled datasets that are freely available [14].

In the extreme case, we do not know the label of any single datapoint. 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 datapoints as **unsupervised ML methods**. We discuss some of the most important unsupervised ML methods in Chaps. 8 and 9).

As discussed next, many ML methods aim at constructing (or finding) a "good" predictor $h : \mathcal{X} \rightarrow \mathcal{Y}$ which takes the features $\mathbf{x} \in \mathcal{X}$ of a datapoint 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 datapoints 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 datapoint, it can be instructive to generate a scatterplot as shown in Fig. 1.2. A scatterplot depicts the datapoints $\mathbf{z}^{(i)} = (x^{(i)}, y^{(i)})$ 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 Fig. 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 Chap. 9). These methods transform high-dimensional datapoints, having billions of raw features, to three or two new features. These new features can then be used as the coordinates of the datapoints in a scatterplot.

### 2.1.4 Probabilistic Models for Data

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

One of the most basic examples of a probabilistic model for datapoints in ML is the i.i.d. assumption. This assumption interprets datapoints $x^{(1)}, \ldots, x^{(m)}$ as realizations

of statistically independent random variables 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 random variables 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 datapoints. 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 actually observing the available data points is maximized [15–17].

Two of the most basic and widely used parameters of a probability distribution $p(x)$ are the expected value or mean [18]

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

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 [19, Chap. 2.3.4]).

Most of the ML methods discussed in this book are motivated by the i.i.d. assumption. It is important to note that this i.i.d. assumption is merely a modelling assumption. There is no means to verify if an arbitrary set of data points are "exactly" realizations of i.i.d. random variables. However, there are principled statistical methods (hypothesis tests) if a given set of data point can be well approximated as realizations of i.i.d. random variables [20]. The only way to ensure the i.i.d. assumption is to 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 [21].

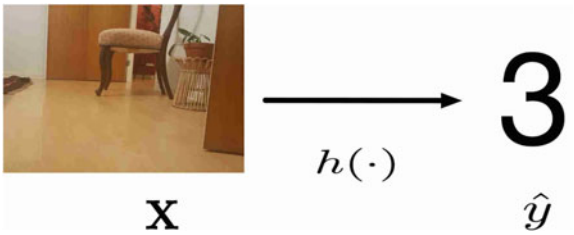



**Fig. 2.6** A hypothesis (predictor) $h$ maps features $\mathbf{x} \in \mathcal{X}$, of an on-board camera snapshot, to the prediction $\hat{y} = h(\mathbf{x}) \in \mathcal{Y}$ for the coordinate of the current location of a cleaning robot. ML methods use data to learn predictors $h$ such that $\hat{y}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

## 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 informal principle of most (if not every) ML method is to learn a hypothesis map $h : \mathcal{X} \rightarrow \mathcal{Y}$ 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 Sect. 2.3. The second issue is then solved in Chap. 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 Fig. 2.6. and Fig. 2.7). It seem natural to refer to a hypothesis map as a predictor map since it is used to compute predictions for the label.

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

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

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

**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

| feature $x$ | prediction $h(x)$ |
|---|---|
| 0 | 0 |
| 1/10 | 10 |
| 2/10 | 3 |
| ⋮ | ⋮ |
| 1 | 22.3 |

In principle, ML methods could use any possible map $h : \mathcal{X} \rightarrow \mathcal{Y}$ 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 search 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 [22].

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 Fig. 2.10, ML methods typically use a hypothesis space $\mathcal{H}$ that is a tiny subset of $\mathcal{Y}^{\mathcal{X}}$. Similar to the features and labels used to characterize data points, also the hypothesis space underlying a ML method is a design choice. 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 available computational infrastructure 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 deep neural networks.

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.

[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 $4^5$.

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 $\hat{h} \in \mathcal{H}$. 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 ($y \in \{\text{cat, no cat}\}$) 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 Sect. 2.3 and Chap. 4). This requirement implies also that the maps $h(\mathbf{x})$ contained in $\mathcal{H}$ can be evaluated (computed) efficiently [23]. Another important reason for using a hypothesis space $\mathcal{H}$ that is not too large is to avoid overfitting (see Chap. 7). If the hypothesis space $\mathcal{H}$ is too large, then just by luck we might 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 [24]. To take advantage of such computational infrastructure, many ML methods use the hypothesis space

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

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

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

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 (or output) $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{x}^T \mathbf{w} \in \mathbb{R}$. For $n = 1$ the feature vector reduces

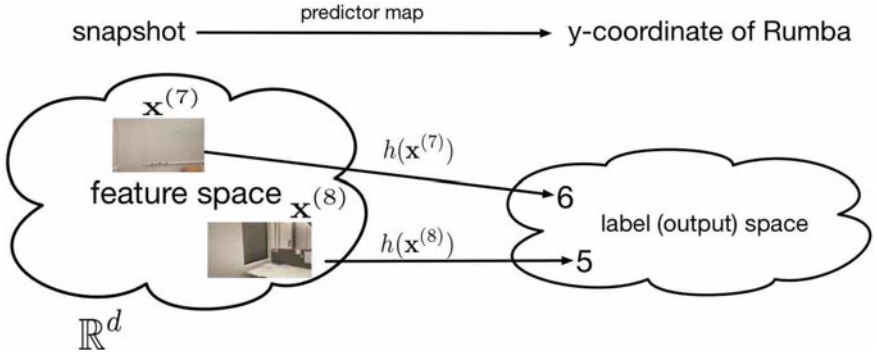



**Fig. 2.7** A hypothesis $h : \mathcal{X} \to \mathcal{Y}$ takes the feature vector $\mathbf{x}^{(t)} \in \mathcal{X}$ (e.g., representing the snapshot taken by Rumba at time t) as input and outputs a predicted label $\hat{y}^{(t)} = h(\mathbf{x}^{(t)})$ (e.g., the predicted $y$-coordinate of Rumba at time $t$). A key problem studied within ML is how to automatically learn a good (accurate) predictor $h$ such that $y^{(t)} \approx h(\mathbf{x}^{(t)})$

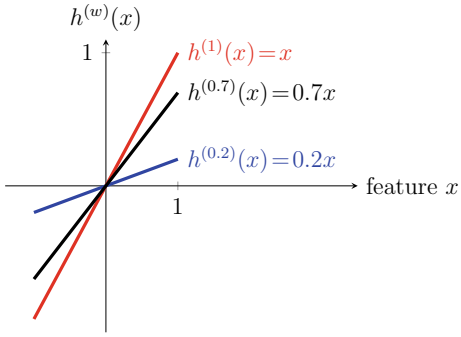



**Fig. 2.8** Three particular members of the hypothesis space $\mathcal{H} = \{h^{(w)} : \mathbb{R} \to \mathbb{R}, h^{(w)}(x) = w \cdot x\}$ which consists of all linear functions of the scalar feature $x$. We can parametrize this hypothesis space conveniently using the weight $w \in \mathbb{R}$ as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

a single feature $x$ and the hypothesis space (2.4) consists of all maps $h^{(w)}(x) = wx$ with some weight $w \in \mathbb{R}$ (see Fig. 2.8).

The elements of the hypothesis space $\mathcal{H}$ in (2.4) are parameterized by the weight vector $\mathbf{w} \in \mathbb{R}^n$. Each map $h^{(\mathbf{w})} \in \mathcal{H}$ is fully specified by the weight 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 weight 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 us to efficiently search over this space. Chapter 5 discusses methods that use the concept of gradients to implement an efficient search for good weights $\mathbf{w} \in \mathbb{R}^n$.

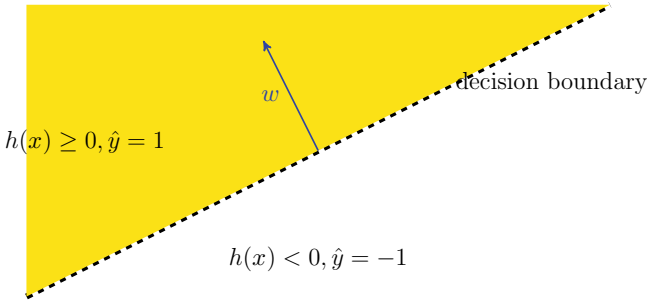



**Fig. 2.9** A hypothesis $h : \mathcal{X} \to \mathcal{Y}$ for a binary classification problem, with label space $\mathcal{Y} = \{-1, 1\}$ and feature space $\mathcal{X} = \mathbb{R}^2$, can be represented conveniently via the decision boundary (dashed line) which separates all feature vectors $\mathbf{x}$ 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

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 Sect. 3.6), the support vector machine (see Sect. 3.7) and the naive Bayes classifier (see Sect. 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 representing images of animals. The map that relates the pixel intensities of image to the label indicating if it is a cat image is highly non-linear. For such applications, the hypothesis space (2.4) is not suitable as it only contains linear maps. The second main example for a parametrized hypothesis space studied in this book also contains non-linear maps. This parametrized hypothesis space is obtained from a parametrized signal flow diagram which is referred to as an artificial neural network. Section 3.11 will discuss the construction of non-linear parametrized hypothesis spaces using an artificial neural network.

**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 $\mathbf{z} = \Phi(\mathbf{x})$. The transformed features are obtained by applying a feature map $\Phi(\cdot)$ ot the original features $\mathbf{x}$. This upgraded hypothesis space $\mathcal{H}'$ consists of all concatenations of the feature map $\Phi$ and some hypothesis $h \in \mathcal{H}$,

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

The construction (2.6) used for arbitrary combinations of a feature map $\Phi(\cdot)$ and a "base" hypothesis space $\mathcal{H}$. The only requirement is that the output of the feature map can be used as input for a hypothesis $h \in \mathcal{H}$. More formally, the range of the feature map 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 Sect. 3.2), Gaussian basis regression (see Sect. 3.5) and the important family of kernal methods (see Sect. 3.9). The feature map in (2.6) might also be obtained from clustering or feature learning methods (see Sects. 8.4 and 9.2.1).

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 $\Phi(\mathbf{x})$. 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 feature vector $\mathbf{x}$ to the predicted label $\hat{y}$,

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

**Non-Numeric Features.** The hypothesis space (2.4) can only be used for datapoints 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 [25, Chap. 12] for more details). Moreover, Sect. 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 datapoints represented by $100 \times 10 = 1000$ black-and-white pixels and characterized by a binary label $y \in \{0, 1\}$. We can model such datapoints 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 $d_{\rm eff}(\mathcal{H})$.

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 $d_{\rm eff}(\mathcal{H})$ 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 [26]. The Vapnik–Chervonenkis (VC) dimension is maybe the most widely used concept for measuring the size of infinite hypothesis spaces [19, 26–28]. However, the precise definition of the Vapnik–Chervonenkis (VC) dimension are beyond the scope of this book. Moreover, the effective dimension captures most of the relevant properties of the Vapnik–Chervonenkis (VC) dimension for our purposes. For a precise definition of the Vapnik–Chervonenkis (VC) dimension and discussion of its applications in ML we refer to [27].

Let us illustrate our concept 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 $\mathcal{D} = \{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\}$ consisting of $m$ data points. We refer to this number also as the sample size of the dataset. 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 assume that data points are realizations of realizations of continuous i.i.d. random variables with a common probability density function. Under this assumption, the 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 rank with probability one. Basic results of linear algebra allow to show that the data points in $\mathcal{D}$ can be perfectly fit by a linear map $h \in \mathcal{H}^{(n)}$ as long as $m \leq n$. As soon as the number $m$ of data points is not strictly larger than the number of features characterizing each data point, i.e., $m \leq n$, we can find (with probability one) a weight vector $\widehat{\mathbf{w}}$ such that $y^{(i)} = \widehat{\mathbf{w}}^T\mathbf{x}^{(i)}$ for all $i = 1, \ldots, m$. The effective dimension of the linear hypothesis space $\mathcal{H}^{(n)}$ is therefore $D = n$.

As a second example, consider the hypothesis space $\mathcal{H}^{(n)}_{\rm poly}$ 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

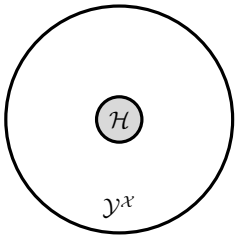



**Fig. 2.10** The hypothesis space $\mathcal{H}$ is a (typically very small) subset of the (typically very large) set $\mathcal{Y}^{\mathcal{X}}$ of all possible maps from feature space $\mathcal{X}$ into the label space $\mathcal{Y}$

the hypothesis space $\mathcal{H}^{(n)}_{\text{poly}}$ is $D = n$. Section 3.2 discusses polynomial regression in more detail.

## 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

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

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

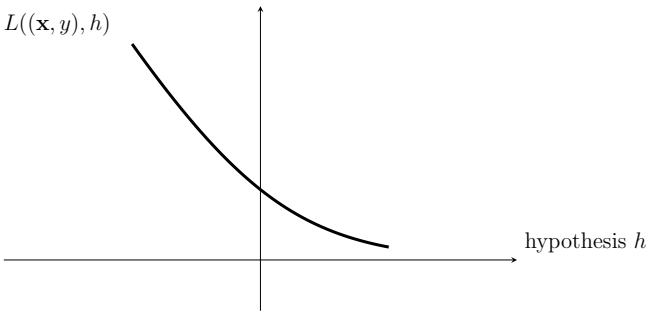



**Fig. 2.11** Some loss function $L((\mathbf{x}, y), h)$ for a fixed data point, with features $\mathbf{x}$ and label $y$, and varying hypothesis $h$. ML methods try to find (learn) a hypothesis that incurs minimum loss

The loss value $L((\mathbf{x}, y), h)$ quantifies the discrepancy between the true label $y$ and the predicted label $h(\mathbf{x})$. A small (close to zero) value $L((\mathbf{x}, y), h)$ indicates a low discrepancy between predicted label and true label of a data point. Figure 2.11 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 $L((\mathbf{x}, y), h)$ for any data point (see Chap. 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 Sects. 2.3.1 and 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 hypothesis space parametrized by a weight vector and a loss function that is a convex and differentiable (smooth) function of the weight vector. In this case, searching for a hypothesis with small loss can be done efficiently using the gradient-based methods discussed in Chap. 5. The minimization of a loss function that is either non-convex or non-differentiable is typically 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 of loss function should also take into account statistical aspects. Some loss functions result in ML methods that are more robust against outliers (see Sects. 3.3 and 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 Sect. 2.3.2). Thus, the average $0/1$ loss can be interpreted as a misclassification (or error) rate. However, using the average $0/1$ loss to learn an accurate hypothesis results in computationally challenging problems. Section 2.3.2 introduces the logistic loss as a computationally attractive alternative choice for the loss function in binary classification 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 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 Chap. 4) and for its final evaluation. Figure 2.12 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.

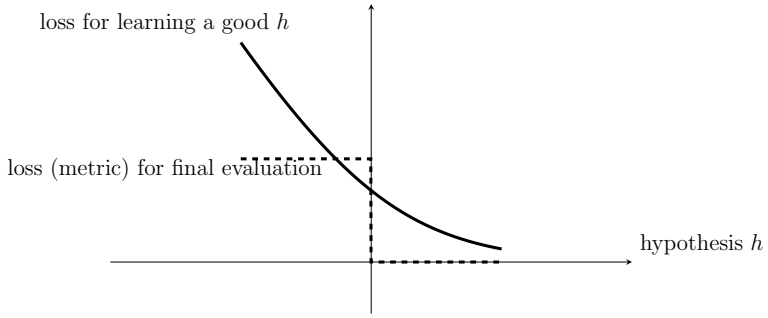



**Fig. 2.12** Two different loss functions for a given data point and varying hypothesis $h$. One loss function (solid curve) is used to learn a good hypothesis by minimizing the loss. Another loss function (dashed curve) is used for the final performance evaluation of the learnt hypothesis. The loss function used for the final performance evaluation is referred to as a metric

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 Chap. 4. The logistic loss is appealing for this purpose as it allows to efficient gradient-based methods (see Chap. 5) to search for an accurate hypothesis. 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.

### *2.3.1 Loss Functions for Numeric Labels*

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

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

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 $L(y, \hat{y})$ for any loss function that depends on the features $\mathbf{x}$ only via the predicted label $\hat{y} = h(\mathbf{x})$. Figure 2.13 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

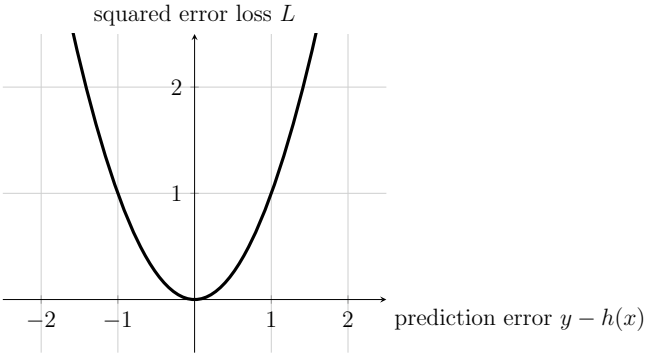



**Fig. 2.13** A widely used choice for the loss function in regression problems (with data points having numeric labels) is the squared error loss (2.8). Note that, for a given hypothesis $h$, we can evaluate the squared error loss only if we know the features $\mathbf{x}$ and the label $y$ of the data point

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 Chap. 5). The squared error loss also has a useful interpretation in terms of a probabilistic model for the features and labels. Minimizing the squared error loss is equivalent to maximum likelihood estimation within a linear Gaussian model [28, Sect. 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 few outliers in the training set (see Sect. 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 assume that labels values are $\mathcal{Y} = \{-1, 1\}$. Classification methods aim at learning a classifier that maps the features $\mathbf{x}$ of a data point to a predicted label $\hat{y} \in \mathcal{Y}$.

We implement a classifier by thresholding the value $h(\mathbf{x}) \in \mathbb{R}$ of a hypothesis 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 classification result, i.e., the predicted label $\hat{y}$, we interpret the absolute value $|h(\mathbf{x})|$ as the confidence in this classification.

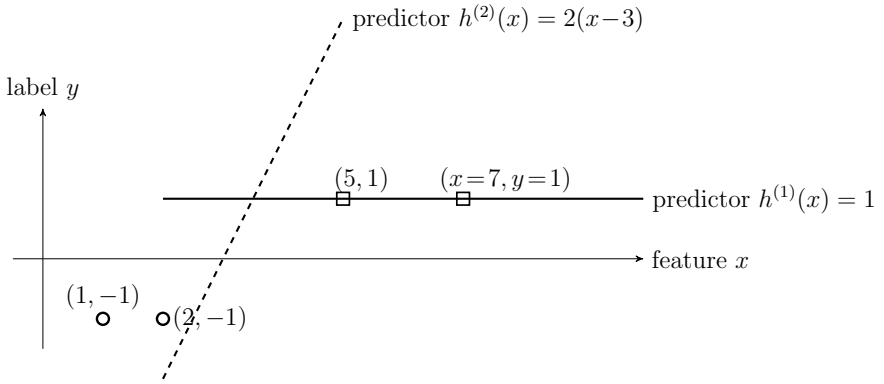



**Fig. 2.14** A training set consisting of four data points with binary labels $\hat{y}^{(i)} \in \{-1, 1\}$. Minimizing the squared error loss (2.8) would prefer the (poor) classifier $h^{(1)}$ over the (reasonable) classifier $h^{(2)}$

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 2.14 illustrates how the squared error loss of a hypothesis can be misleading in a binary classification problem.

Figure 2.14 depicts a dataset 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 hypotheses $h^{(1)}(x)$ and $h^{(2)}(x)$ that can be used for classifying data points. The classifications $\hat{y}$ obtained with the hypothesis $h^{(2)}(x)$ would perfectly match the labels of the four training data points since $h^{(2)}\big(x^{(i)}\big) \geq 0$ 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 $y = -1$. Thus, based on the training data, we would prefer using $h^{(2)}(x)$ over $h^{(1)}$ to classify data points. However, the squared error loss incurred by the (reasonable) classifier $h^{(2)}$ is much larger than the squared error loss incurred by the (poor) classifier $h^{(1)}$. The squared error loss is typically a bad choice for assessing the quality of a hypothesis map that is used for classifying data points into different categories.

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

We now discuss some loss functions which have proven useful for assessing the quality of a hypothesis 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, i.e., when the label space is $\mathcal{Y} = \{-1, 1\}$. These formulas need to modified accordingly if one prefers to use different label values for a binary classification

problem. 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"$\}$.

The first loss function that we discuss is direct formalization of the natural requirement for a hypothesis to result on correct classifications, i.e., $\hat{y} = y$ for any data point. This suggests to learn a hypothesis $h(\mathbf{x})$ by minimizing the $0/1$ loss

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

Figure 2.15 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. random variables 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. random variables,

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

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 [18, Section 1]. We can apply the law of large numbers since the loss values $L((\mathbf{x}^{(i)}, y^{(i)}), h)$ are realizations of i.i.d. random variables. The average $0/1$ loss on the left-hand side of (2.10) is referred to as the **accuracy** of the hypothesis $h$.

In view of (2.10), the $0/1$ loss seems a very natural choice for assessing the quality of a classifier if our goal is to enforce correct classification ($\hat{y} = y$). This appealing statistical property of the $0/1$ loss comes at the cost of high computational complexity. Indeed, for a given data point $(\mathbf{x}, y)$, the $0/1$ loss (2.9) is neither convex nor differentiable when viewed as a function of the classifier $h$. Thus, using the $0/1$ loss for binary classification problems typically involves advanced optimization methods for solving the resulting learning problem (see Sect. 3.8).

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

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

Figure 2.15 depicts the hinge loss (2.11) as a function of the hypothesis $h(\mathbf{x})$. While the hinge loss avoids the non-convexity of the $0/1$ loss it still is a non-differentiable function of the classifier $h$. Non-differentiable loss functions are typically harder to minimize, implying a higher computational complexity of the ML method using such a loss.

Section 3.6 discusses the logistic loss which is a differentiable loss function that is useful for classification problems. The logistic loss

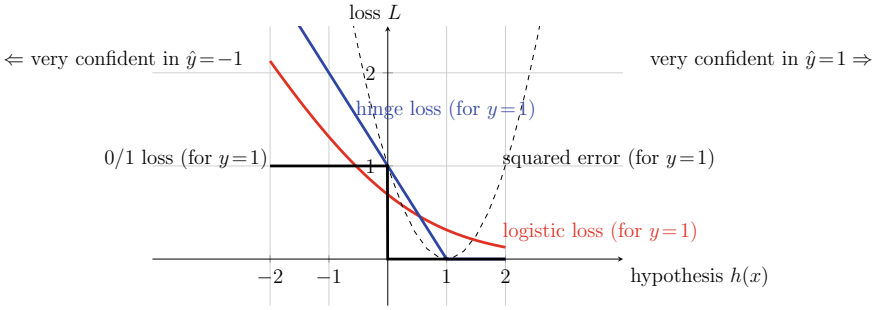



**Fig. 2.15** The solid curves depict three widely-used loss functions for binary classification problems. A data point is classified as $\hat{y} = 1$ if $h(x) \geq 0$ and classified as $\hat{y} = -1$ if $h(x) < 0$. We can interpret the absolute value $|h(x)|$ as the confidence in the classification. The more confident we are in a correct classification ($\hat{y}=1$), i.e., the more positive $h(x)$, the smaller the loss. Note that each of the three loss functions for binary classification tends monotonically to 0 for increasing $h(x)$. The dashed curve depicts the squared error loss (2.8), which increases for increasing $h(x)$

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

is used within logistic regression to measure the usefulness of a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

Consider a specific data point with the feature vector $\mathbf{x} \in \mathbb{R}^n$ and a binary label $y \in \{-1, 1\}$. We use a linear hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, with some weight vector $\mathbf{w} \in \mathbb{R}^n$, to predict the label based on the features $\mathbf{x}$ according to $\hat{y} = 1$ if $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} > 0$ and $\hat{y} = -1$ otherwise. Then, both the hinge loss (2.11) and the logistic loss (2.12) are **convex functions** of the weight vector $\mathbf{w} \in \mathbb{R}^n$. The logistic loss (2.12) depends smoothly on $\mathbf{w}$. It is a differentiable function in the sense of allowing to define a gradient with respect to $w$. In contrast, the hinge loss (2.11) is nonsmooth which makes it more difficult to minimize [29, Chap. 3].

ML methods that use the convex and differentiable logistic loss function, such as logistic regression in Sect. 3.6, can apply simple gradient-based methods such as gradient descent (GD) to minimize the average loss (see Chap. 5). In contrast, we cannot use gradient-based methods to minimize the hinge loss since it is not differentiable. However, we can apply a generalization of GD which is known as subgradient descent [30] Subgradient descent is obtained from GD by generalizing the concept of a gradient to that of a subgradient.

### *2.3.3 Loss Functions for Ordinal Label Values*

There are also loss functions particularly suited for predicting ordinal label values (see Sect. 2.1). Consider data points representing areal images of rectangular areas of size 1 km by 1 km. We characterize each data point (rectangular area) by the

feature vector $\mathbf{x}$ obtained by stacking the RGB values of each image pixel (see Fig. 2.4). 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 area 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 $y = 2$. It might be useful to take the ordering of label values into account when evaluating the quality of the predictions obtained by a hypothesis $h(\mathbf{x})$.

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 from 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

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

### 2.3.4 Empirical Risk

The basic idea of ML methods (including those discussed in Chap. 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. random variables with a common probability distribution $p(\mathbf{x}, y)$. Given such a probabilistic model, it seems natural to measure the quality of a hypothesis by the expected loss or Bayes risk [15]

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

The Bayes risk is the expected value of the loss $L((\mathbf{x}, y), h)$ 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 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,

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

A hypothesis that solves (2.15), i.e., that achieves the minimum possible Bayes risk, is referred to as a Bayes estimator [15, Chap. 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 [15, 31].

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 $\mathcal{D} = \big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)$. In particular, we define the empirical risk of a hypothesis $h \in \mathcal{H}$ for a dataset $\mathcal{D}$ as

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

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. random variables whose common distribution is $p(\mathbf{x}, y)$, basic results of probability theory tell us that

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

The approximation error in (2.17) can be quantified precisely by some of the most basic results of probability theory. These results are referred to as the law of large numbers.

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 $y$.

#### 2.3.4.1 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 label values $p, q \in \{1, \ldots, k\}$, we define the loss

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

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

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

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

#### 2.3.4.2 Precision, Recall and F-Measure

Consider an object detection application where data points represent 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 [32]

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

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,

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

The $F$ measure (2.21) is the harmonic mean [33] of the precision and recall of a hypothesis $h$. It is a special case of the $F_\beta$-score

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

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 or **experts**. The quality of a hypothesis $h$ can then be measured via the difference between the loss incurred by its predictions $h(\mathbf{x})$ and the loss incurred by the predictions of the experts [34]. 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 (following) the prediction of the expert. The goal of regret minimization is to learn a hypothesis with a small regret compared to all considered experts.

The concept of regret minimization is useful when we do not make any probabilistic assumptions (see Sect. 2.1.4) about the data. Without a probabilistic model we cannot use the Bayes risk, which is the risk of the Bayes estimator, as a benchmark.

Regret minimization techniques can be designed and analyzed without any such probabilistic model for the data [35]. This approach replaces the Bayes risk with the regret relative to given reference predictors (experts) as the benchmark.

### 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 [35, 36].

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 datapoints 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

The main theme of the book is that ML methods are obtained by different combinations of data, model and loss. We will discuss some key principles behind these combinates 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 datapoints

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

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 Chap. 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. In other words, given any datapoint 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,

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

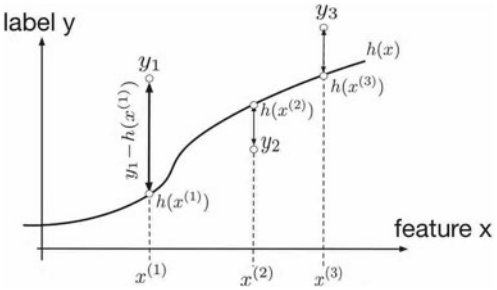



**Fig. 2.16** We can evaluate the quality of a particular predictor $h \in \mathcal{H}$ by measuring the prediction error $y - h(x)$ obtained for a labeled datapoint $(x, y)$

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 Fig. 2.16).

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

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

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 datapoints (2.23).

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

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

The optimal weights $w_0', w_1'$ are characterized by the **zero-gradient condition**,[2]

$$\frac{\partial f(w_0', w_1')}{\partial w_0} = 0, \text{ and } \frac{\partial f(w_0', w_1')}{\partial w_1} = 0. \tag{2.27}$$

[2] A necessary and sufficient condition for $\mathbf{w}'$ to minimize a convex differentiable function $f(\mathbf{w})$ is $\nabla f(\mathbf{w}') = \mathbf{0}$ [37, Sec. 4.2.3].

Inserting (2.25) into (2.27) and by using basic rules for calculating derivatives, we obtain the following optimality conditions

$$(1/m)\sum_{i=1}^{m}\big(y^{(i)}-(w_1'x^{(i)}+w_0')\big)=0, \text{ and } (1/m)\sum_{i=1}^{m}x^{(i)}\big(y^{(i)}-(w_1'x^{(i)}+w_0')\big)=0. \quad (2.28)$$

Any weights $w_0'$, $w_1'$ that satisfy (2.28) define a predictor $h^{(w_0',w_1')}=w_1'x+w_0'$ that is optimal in the sense of incurring minimum training error,

$$f(w_0',w_1')=\min_{w_0,w_1\in\mathbb{R}} f(w_0,w_1).$$

We find it convenient to rewrite the optimality condition (2.28) using matrices and vectors. To this end, we first rewrite the predictor (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 training datapoints (2.23) into the feature matrix and label vector,

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

We can then reformulate (2.28) as

$$\mathbf{X}^T\big(\mathbf{y}-\mathbf{X}\mathbf{w}'\big)=\mathbf{0}. \quad (2.30)$$

The entries of any weight vector $\mathbf{w}'=\big(w_0',w_1'\big)$ that satisfies (2.30) are solutions to (2.28).

## 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 $h:\mathbb{R}\rightarrow\mathbb{R}$ 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 training 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 Fig. 1.3).

**Exercise 2.2 Temperature Data** Consider a dataset of daily air temperatures $x^{(1)},\ldots,x^{(m)}$ measured at the observation station Utsjoki Nuorgam between 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 measure don 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 parameters

- 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 $D := \sqrt{(1/m) \sum_{i=1,\ldots,m} \left(x^{(i)} - C\right)^2}$

**Exercise 2.3 Deep Learning on Raspberry PI** Consider the tiny desktop computer "RaspberryPI" equipped with a total of 8 Gigabytes memory [38]. On that computer, we want implement a ML algorithm that learns a hypothesis map that is represented by a deep neural network involving $n = 10^6$ numeric weights (or parameters). Each weight 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 = $10^9$Bytes.)

**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 aggreation 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 [39]. 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 $\mathbf{x} \in \mathbb{R}^n$ if its entries are the signal amplitudes of a 20-second long recording which is sampled at a rate of 44 kHz?

**Exercise 2.6 Multilabel Prediction.** Consider datapoints 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?

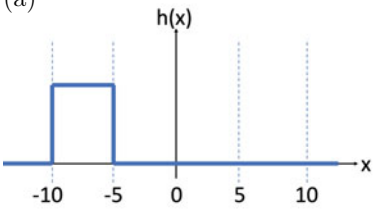
**Exercise 2.7 Average Squared Error Loss as Quadratic Form** Consider the hypothesis space constituted by all linear maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with some weight vector $\mathbf{w} \in \mathbb{R}^n$. We try to find the best linear map by minimizing the average squared error loss (the 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 feature vectors and labels of the training data?

**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 parameterized 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 datapoints.

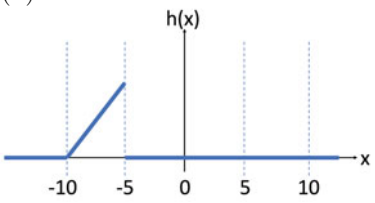
**Exercise 2.9 Dummy Feature Instead of Intercept** Show that any hypothesis map of the form $h(x) = w_1 x + w0$ can be obtained as concatenation of a feature map $\Phi : x \mapsto \mathbf{z}$ with a map $\tilde{h}(\mathbf{z}) := \widetilde{\mathbf{w}}^T\mathbf{z}$ with some weight vector $\widetilde{\mathbf{w}} \in \mathbb{R}^2$.

**Exercise 2.10 Approximate Non-Linear Maps Using Indicator Functions for Feature Maps.** Consider an ML application generating datapoints 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 $\mathbf{z} = (\phi_1(x), \phi_2(x), \phi_3(x), \phi_4(x))$. 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 construct a hypothesis space $\mathcal{H}_1$ by all maps of the form $\mathbf{w}^T\mathbf{z}$. Note that the map is a function of the feature $x$ since the feature vector $\mathbf{z}$ is a function of $x$. Which of the following predictor maps belong to $\mathcal{H}_1$?

(a)




(b)




**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-Parameterization** Consider datapoints 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

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

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

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

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 $j \in \{1, 2, 3\}$, $h^{(j)}(x) = 0$ for all $x^2 \leq j$ and $h^{(j)}(x) = j$ otherwise. Can you tell which of these hypothesis is optimal in the sense of having smallest average squared error loss on the three (training) datapoints $(x = 1/10, y = 3)$, $(0, 0)$ and $(1, -1)$.

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

**Exercise 2.16 Intercept Term** Linear regression methods model the relation between the label $y$ and feature $x$ of a datapoint 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 weight $w_0$ 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 Maximum Hypothesis space** Consider datapoints 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 datapoint and deliver a real-valued prediction $\hat{y} = h(x)$ ?

**Exercise 2.19 A Large but Finite Hypothesis space** Consider datapoints whose features are $10 \times 10$ black-and-white (bw) pixel images. Each datapoint is also characterized by a binary label $y \in \{0, 1\}$. Consider the hypothesis space which is constituted by all maps that take a 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 $m$ datapoints with feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and numeric labels $y^{(1)}, \ldots, y^{(m)}$. The feature vectors and label values of the training set are arbitrary except that we assume the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots\big)$ 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., $y^{(1)} = h\big(\mathbf{x}^{(1)}\big), \ldots, y^{(m)} = h\big(\mathbf{x}^{(m)}\big)$.

# References


1. K. Abayomi, A. Gelman, M.A. Levy, Diagnostics for multivariate imputations. Journal of The Royal Statistical Society Series C-applied Statistics **57**, 273–291 (2008)
2. W. Rudin, *Principles of Mathematical Analysis*, 3rd edn. (McGraw-Hill, New York, 1976)
3. P. Bühlmann, S. van de Geer, *Statistics for High-Dimensional Data* (Springer, New York, 2011)
4. M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint* (Cambridge University Press, Cambridge, 2019)
5. R. Vidal, Subspace clustering. *IEEE Signal Processing Magazine*, March 2011
6. F. Barata, K. Kipfer, M. Weber, P. Tinschert, E. Fleisch, and T. Kowatsch, Towards device-agnostic mobile cough detection with convolutional neural networks, in *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, pp. 1–11 (IEEE, New York, 2019)
7. B. Boashash (ed.), *Time Frequency Signal Analysis and Processing: A Comprehensive Reference* (Elsevier, Amsterdam, The Netherlands, 2003)
8. S.G. Mallat, *A Wavelet Tour of Signal Processing - The Sparse Way*, 3rd edn. (Academic Press, San Diego, CA, 2009)
9. S. Smoliski, K. Radtke, Spatial prediction of demersal fish diversity in the Baltic sea: comparison of machine learning and regression-based techniques. ICES Journal of Marine Science **74**(1), 102–111 (2017)
10. S. Carrazza, Machine learning challenges in theoretical HEP (2018)
11. M. Gao, H. Igata, A. Takeuchi, K. Sato, Y. Ikegaya, Machine learning-based prediction of adverse drug effects: An example of seizure-inducing compounds. Journal of Pharmacological Sciences **133**(2), 70–78 (2017)
12. K. Mortensen, T. Hughes, Comparing amazon's mechanical Turk platform to conventional data collection methods in the health and medical research literature. J. Gen. Intern Med. **33**(4), 533–538 (2018)
13. A. Halevy, P. Norvig, F. Pereira, *The unreasonable effectiveness of data* (IEEE Intelligent Systems, New York, 2009)
14. P. Koehn, Europarl: A parallel corpus for statistical machine translation, in *The 10th Machine Translation Summit*, pp. 79–86 (AAMT, 2005)
15. E.L. Lehmann, G. Casella, *Theory of Point Estimation*, 2nd edn. (Springer, New York, 1998)
16. S.M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory* (Prentice Hall, Englewood Cliffs, NJ, 1993)
17. D. Bertsekas, J. Tsitsiklis, *Introduction to Probability*, 2nd edn. (Athena Scientific, Singapore, 2008)
18. P. Billingsley, *Probability and Measure*, 3rd edn. (Wiley, New York, 1995)
19. C.M. Bishop, *Pattern Recognition and Machine Learning* (Springer, Berlin, 2006)
20. H. Lütkepohl, *New Introduction to Multiple Time Series Analysis* (Springer, New York, 2005)
21. B. Efron, R. Tibshirani, Improvements on cross-validation: The 632+ bootstrap method. Journal of the American Statistical Association **92**(438), 548–560 (1997)
22. P. Halmos, *Naive Set Theory* (Springer, Berlin, 1974)
23. P. Austin, P. Kaski, and K. Kubjas, Tensor network complexity of multilinear maps (2018)
24. F. Pedregosa, Scikit-learn: Machine learning in python. Journal of Machine Learning Research **12**(85), 2825–2830 (2011)
25. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016)
26. V.N. Vapnik, *The Nature of Statistical Learning Theory* (Springer, Berlin, 1999)
27. S. Shalev-Shwartz, S. Ben-David, *Understanding Machine Learning-From Theory to Algorithms* (Cambridge University Press, New York, 2014)
28. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning* Springer Series in Statistics. (Springer, New York, 2001)
29. Y. Nesterov, *Introductory Lectures on Convex Optimization, Vol. 87 of Applied Optimization* (Kluwer Academic Publishers, Boston, 2004)
30. S. Bubeck, Convex optimization: Algorithms and complexity. Foundations and Trends in Machine Learning **8**(3–4), 231–357 (2015)

31. M.J. Wainwright, M.I. Jordan, *Graphical Models, Exponential Families, and Variational Inference, Foundations and Trends in Machine Learning*, vol. 1 (Now Publishers, Hanover, MA, 2008)
32. R. Baeza-Yates, B. Ribeiro-Neto, *Modern Information Retrieval* (ACM Press, New York, 1999)
33. M. Abramowitz, I.A. Stegun (eds.), *Handbook of Mathematical Functions* (Dover, New York, 1965)
34. E. Hazan, *Introduction to Online Convex Optimization* (Now Publishers Inc., Hanover, MA, 2016)
35. N. Cesa-Bianchi, G. Lugosi, *Prediction, Learning, and Games* (Cambridge University Press, New York, 2006)
36. R. Sutton, A. Barto, *Reinforcement Learning: An Introduction*, 2nd edn. (MIT Press, Cambridge, MA, 2018)
37. S. Boyd, L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, UK, 2004)
38. O. Dürr, Y. Pauchard, D. Browarnik, R. Axthelm, and M. Loeser. Deep learning on a raspberry pi for real time face recognition (2015)
39. A. Wang, An industrial-strength audio search algorithm, in *International Symposium on Music Information Retrieval* (2003)