Lecture Notes for

MVE137
Probability and Statistical Learning Using Python

Alexandre Graell i Amat

Chalmers University of Technology
Department of Electrical Engineering
`alexandre.graell@chalmers.se`
https://sites.google.com/site/agraellamat/

September 25, 2022

Contents

Chapter 1

Introduction

The main goal of statistical learning is to provide a framework for studying the problem of inference from data, i.e., the problem of gaining knowledge, making predictions, making decisions or constructing models from a set of data. In plain words, it provides a set of tools for *learning from* and *understanding* data. Statistical learning is closely related to machine learning—statistical learning provides the theoretical basis for many of today's machine learning algorithms.

More concretely, the goal of statistical learning is that of inferring (or learning) a predictive function—sometimes called model, in particular in machine learning—, such that it can be used to predict the output for new, yet unseen data.

Statistical learning finds applications in areas such as computer vision, speech recognition, bioinformatics. Specific examples of learning problems are predicting apartment prices based on some available data such as square meters, number of rooms, location, orientation south or north, whether the apartment has a balcony or not; image classification, particularly popular in computer vision, where the goal is to predict what class an image belongs to, e.g., a cat or a dog, based on available labeled images; or predicting stock prices based on stock price history, financial statements of the companies, or consumer behaviors. In all of these scenario, we wish to predict an outcome, for example the value of a stock or the numbers of a hand-written postal code, based on a set of inputs (such as diet and clinical measurements). All these examples fall within the category of supervised learning. Supervised learning encompasses the statistical learning processes that exploit available data with known outcomes to guide the learning process. The learning is then based on available labeled data, and the function is chosen in order to minimize a given loss-function, which quantifies the discrepancy between the observed data and the predictions.

In this course, we introduce the basic concepts underpinning supervised learning and discuss linear methods for regression and classification. We further give a first glimpse to learning methods that go beyond linearity.

1.1 Notation

We will use lowercase bold letters to denote vectors, uppercase bold letters to denote matrices, and calligraphic letters to denote sets, e.g., $\boldsymbol{x}$, $\boldsymbol{X}$, and $\mathcal{X}$. Unless otherwise stated, vectors will be column vectors. The set of real numbers is denoted by $\mathbb{R}$ and the set of natural numbers by $\mathbb{N}$. We will use $(\cdot)^\mathsf{T}$ to represent the transpose of a vector or matrix. $\mathbb{1}\{\cdot\}$ will denote an indicator function, i.e., the function returns 1 if the condition inside the braces is fulfilled and 0 otherwise. We will use sansserif font to denote random variables, e.g., x, $\boldsymbol{\mathsf{x}}$, and $\boldsymbol{\mathsf{X}}$ denotes a random variable, a random vector, and a random matrix, respectively.

The distribution of a random variable x, either probability mass function (pmf) for a discrete random variable or probability density function (pdf) for continuous random variables, is denoted as p_{x}, $p_{\mathsf{x}}(x)$, or $p(x)$. The notation $\mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}[\cdot]$ and $\mathbb{E}_{\mathsf{x}}[\cdot]$ indicates the expectation of the argument with respect to the distribution of the RV $\mathsf{x} \sim p(x)$. Accordingly, we will also write $\mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}|\mathsf{y}}}[\cdot|y]$ or $\mathbb{E}_{\mathsf{x}|\mathsf{y}}[\cdot|y]$ for the conditional expectation with respect to the distribution $p_{\mathsf{x}|\mathsf{y}=y}$. When clear from the context, the distribution over which the expectation is computed may be omitted. Further,

$$\begin{aligned}\mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}[f(x)] &= \int f(x)p(x)\mathrm{d}x \\ \mathrm{Var}_{\mathsf{x}\sim p_{\mathsf{x}}}[f(x)] &= \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}\left[(f(x) - \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}[f(x)])^2\right]\end{aligned}$$

$\boldsymbol{\mathsf{x}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ indicates that random vector $\boldsymbol{\mathsf{x}}$ is distributed according to a multivariate Gaussian pdf with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

Chapter 2

Supervised learning

Statistical learning can be classified as supervised or unsupervised. In this course we focus on supervised learning.

Supervised learning is a subcategory of statistical (and machine) learning defined by using labeled datasets to train a statistical model for accurately predicting outcomes.

The labeled dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$, called *training set*, comprises N output variables $y_1, \dots, y_N$ (called labels or responses), which are the variables we would like to predict (e.g., the price of an apartment or a stock) and, for each output y_i, a vector of d inputs $\boldsymbol{x}_i = (x_{i,1}, \dots, x_{i,d})^\mathsf{T}$, known as explanatory variables (as they *explain* or *encode information* about y), features—in machine learning—, or predictors—in statistics. For the example of y being the price of an apartment, $\boldsymbol{x}$ could be a vector of features such as number of square meters, number of rooms, or orientation south or north for an apartment.

Each pair $(\boldsymbol{x}_i, y_i)$ is called a training point or training example. Each training example is thus a pair consisting of an input object (typically a vector) and the corresponding output value.

Based on the training set, supervised learning aims to build a prediction model, or learner, which will be used to predict the outcome y for a newly yet unobserved input data. A good model is one that accurately predicts such an outcome. For example, given some atmospheric measurements today, predict the weather in Gothenburg tomorrow.

Generally, we distinguish between *regression* problems, in which the aim is to predict a continuous variable t and *classification* problems, in which the label t is discrete. An example of a regression task is the prediction of tomorrow's value of a stock t based on historical data $\boldsymbol{x}$.

Regression and classification share many properties, and in particular, both can be viewed as a task in function approximation, as we will see in the following chapters.

2.1 Statistical inference and decision theory

Suppose that we have an input vector $\boldsymbol{x}$ together with a corresponding outcome y. We will assume $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Our goal is to predict y given a new, previously unseen $\boldsymbol{x}$. We will first focus on the regression problem. To be able to perform the learning task, we require some information on the process relating $\boldsymbol{x}$ and y. Without assuming that, for example, $\boldsymbol{x}$ and y are related by a function $y = \tilde{f}(\boldsymbol{x})$ with some properties, such as smoothness, the task of predicting the output y for an unobserved feature $\boldsymbol{x}$ is rendered impossible.

Learning, i.e., predicting y given $\boldsymbol{x}$, can be seen as a statistical inference (or statistical learning) problem.

Treating the input $\boldsymbol{x}$ and the output y as random variables, x and y, respectively, the (unknown) joint probability distribution $p(\boldsymbol{x}, y)$ provides a complete summary of the uncertainty associated with these variables. Determining $p(\boldsymbol{x}, y)$ using a training data set is an example of inference. In machine learning, our interest is usually on the conditional distribution $p(y|\boldsymbol{x})$.

The general inference problem then involves determining the joint distribution $p(y, \boldsymbol{x})$—or the conditional distribution $p(y|\boldsymbol{x})$—, which gives us the most complete probabilistic description of the data. $p(y|\boldsymbol{x})$ and $p(\boldsymbol{x}, y)$ are related via the Bayes' theorem,

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x})}\,.$$

Although the conditional distribution $p(y|\boldsymbol{x})$ can be a very useful and informative quantity, in most practical applications, we must often make a specific prediction for the value of y. This decision step is the subject of decision theory.

We will come back to the distribution $p(y|\boldsymbol{x})$ in Section 2.7. For the moment, we focus our attention on modeling the function $\tilde{f}(\boldsymbol{x})$ which summarizes the dependence of y on $\boldsymbol{x}$ (note that for the moment we assume that the relationship between $\boldsymbol{x}$ and y is deterministic). Ideally, we would like to find the best possible approximation of $\tilde{f}(\boldsymbol{x})$, which we denote by $f(\boldsymbol{x})$, i.e., the optimal predictor $\hat{y}$ of y for previously unseen data $\boldsymbol{x}$. Based on $f(\boldsymbol{x})$, we can then predict the outcome y for $\boldsymbol{x}$ via $\hat{y} = f(\boldsymbol{x})$.

In supervised learning, we are given a training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\} = (\boldsymbol{X}_{N\times p}, \boldsymbol{y}_{N\times 1})$, consisting of N training examples, which we assume that are independent identically distributed (i.i.d.) random variables drawn from a true (and unknown) distribution $p(\boldsymbol{x}, y)$,

$$(\mathsf{x}_i, \mathsf{y}_i) \sim_{i.i.d.} p(\boldsymbol{x}, y), \quad i \in [N]\,.$$

We want to use this data to learn a model $f(\boldsymbol{x})$ that approximates well the true function $\tilde{f}(\boldsymbol{x})$. We call $f(\boldsymbol{x})$ our prediction model, imagining a situation in the future where we will get $\boldsymbol{x}$ only, and then apply $f(\boldsymbol{x})$ to get a prediction $\hat{y} = f(\boldsymbol{x})$. A good prediction model would tend to give $\hat{y} \approx y$.

Thus, the fundamental question that we want to answer is, based on the training set $\mathcal{D}$, which function $f(\boldsymbol{x})$ should we select?[1] In order to perform optimal inference, we start by selecting a nonnegative loss function $\ell(y, \hat{y}) = \ell(y, f(\boldsymbol{x}))$ that defines the cost (also called loss or risk) incurred when the correct value is y while the estimate is $\hat{y}$: The loss function penalizes prediction errors. For linear regression, the most popular loss function is the quadratic loss function (also called squared error loss function)

$$\ell(y, \hat{y}) = (y - \hat{y})^2 = (y - f(\boldsymbol{x}))^2, \tag{2.1}$$

i.e., the squared loss between the outcome y and the prediction $\hat{y} = f(\boldsymbol{x})$ dependent on the variables $\boldsymbol{x}$.

For a given loss function, the function yielding the optimal prediction $\hat{y}(\boldsymbol{x})$ for a new, previously unseen input $\boldsymbol{x}$ is then obtained by minimizing the expected prediction error—also called expected generalization loss or generalization error—,

$$\begin{aligned} L(\hat{y}) &= \mathbb{E}_{\mathsf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[\ell(\mathsf{y}, f(\mathsf{x}))] \qquad (2.2)\\ &= \int\int \ell(\mathsf{y}, f(\boldsymbol{x}))p(\boldsymbol{x}, y)\mathrm{d}\boldsymbol{x}\mathrm{d}y\,. \end{aligned}$$

[1]A secondary goal of the learning is often to learn properties about the true function $\tilde{f}(\boldsymbol{x})$, such as which features in $\boldsymbol{x}$ are actually informative about y, i.e., have a role in f or, more generally, what is the nature of function f, for example, is it linear, or can it be well approximated by linear functions?

Hence, the optimal prediction consists of solving the following optimization problem,

$$\begin{aligned} f^*(\boldsymbol{x}) &= \operatorname*{argmin}_f L(\hat{y}) \\ &= \operatorname*{argmin}_f \mathbb{E}_{\mathsf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[\ell(\mathsf{y}, f(\mathbf{x}))] \,. \end{aligned} \tag{2.3}$$

Applying the law of iterated expectations

$$\mathbb{E}_{(\mathsf{x},\mathsf{y})\sim p_{\mathsf{xy}}}[\cdot] = \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}\left[\mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathsf{x}}}[\cdot|\mathbf{x}]\right],$$

(2.3) can be rewritten as

$$f^*(\boldsymbol{x}) = \operatorname*{argmin}_f \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}\left[\mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathsf{x}}}[\ell(\mathsf{y}, f(\boldsymbol{x}))]\right]. \tag{2.4}$$

The outer expectation averages over the inputs $\boldsymbol{x}$, hence it suffices to minimize the expected prediction error pointwise,[2]

$$f^*(\boldsymbol{x}) = \operatorname*{argmin}_f \mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathsf{x}}}[\ell(\mathsf{y}, f(\boldsymbol{x}))] \tag{2.5}$$

for a given point $\boldsymbol{x}$ (the new, previously unseen data).

Let's particularize the optimal solution for the quadratic loss function (2.1). In this case, the optimal prediction is the conditional expectation

$$\hat{y}^*(x) = f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] \,. \tag{2.6}$$

$f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}]$ is sometimes referred to as the *regression function.*

Let's show this result,

$$\begin{aligned} \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\ell(\mathsf{y}, f(\boldsymbol{x}))|\boldsymbol{x}] &= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\left[(\mathsf{y} - f(\boldsymbol{x}))^2|\boldsymbol{x}\right] \\ &= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\left[(\mathsf{y} - f(\boldsymbol{x}))^2\right] \end{aligned}$$

To find the optimal, we take the derivative an equate to zero,

$$\begin{aligned} \frac{\partial}{\partial f(\boldsymbol{x})}\mathbb{E}_{\mathsf{y}|\mathsf{x}}\left[(\mathsf{y} - f(\boldsymbol{x}))^2\right] &= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\left[\frac{\partial}{\partial f(\boldsymbol{x})}\left((\mathsf{y} - f(\boldsymbol{x}))^2\right)\right] \\ &= \mathbb{E}_{\mathsf{y}|\mathsf{x}}[-2(\mathsf{y} - f(\boldsymbol{x}))] \end{aligned}$$

Now

$$\begin{aligned} \mathbb{E}_{\mathsf{y}|\mathsf{x}}[-2(\mathsf{y} - f(\boldsymbol{x}))] = 0 &\Longrightarrow \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y} - f(\boldsymbol{x})] = 0 \\ &\Longrightarrow \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] - f(\boldsymbol{x}) = 0 \\ &\Longrightarrow f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] \,. \end{aligned}$$

Thus, for a quadratic loss function the best prediction of y at any point $\mathbf{x} = \boldsymbol{x}$ is the conditional mean.

In practice, there is a fundamental problem with this approach. In statistics, the joint probability distribution $p(\boldsymbol{x}, y)$ is assumed to be known, hence can be solved. For most practical learning problems, however, $p(\boldsymbol{x}, y)$ is unknown. Hence, the regression methods that we will discuss in the following can be seen as ways to approximate $\mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}]$.

[2]By conditioning on $\mathbf{x}$, the function f does not depend on $\boldsymbol{x}$ and since $(y - f(\boldsymbol{x}))^2$ is convex, there is a unique solution.

2.2 Basic predictive modeling approaches: least squares linear regression

How do we choose a good function $f(\boldsymbol{x})$ and, how do we effectively use the training set $\mathcal{D}$ to guide this choice?

The space of all possible functions $f(\boldsymbol{x})$ is enormous and hence we need to reduce the search space. We will start from describing two important and fundamental methods for regression. In this section, we introduce least squares linear regression, and we discuss k-nearest neighbors regression in the next section.

To restrict the space of functions $f(\boldsymbol{x})$ and make learning more tractable, linear regression assumes a linear model for $f(\boldsymbol{x})$ in which y can be written as a weighted sum of the elements of $\boldsymbol{x}$. Given a vector of inputs $\boldsymbol{x} = (x_1, x_2, \dots, x_d)^\mathsf{T}$, $f(\boldsymbol{x})$ is given by

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} x_j w_j, \tag{2.7}$$

and we predict the output y as $y = f(\boldsymbol{x})$.

Thus, in linear regression we model the label y as a linear function of the features $x_1, \dots, x_d$, where $w_i \in \mathbb{R}$. Parameter w_0 is known as the intercept, or bias in machine learning.

Despite its simplicity, linear regression remains one of our most important tools in statistical (and machine) learning.

Typically, we want to also include the intercept w_0 directly in our model, so we define

$$\tilde{\boldsymbol{x}} = (1, x_1, \dots, x_d)^\mathsf{T} \quad \text{and} \quad \boldsymbol{w} = (w_0, \dots, w_d)^\mathsf{T}$$

With this, we can rewrite (2.7) as

$$f(\boldsymbol{x}) = \sum_{j=0}^{d} \tilde{x}_j w_j,$$

which can also be written in vector notation as

$$f(\boldsymbol{x}) = \tilde{\boldsymbol{x}}^\mathsf{T} \boldsymbol{w}$$

Linear regression is a parametric model with $d+1$ parameters, $w_0, \dots, w_d$.

For notation simplicity, we will often neglect the tildes and simply write $\boldsymbol{x}$ and

$$f(\boldsymbol{x}) = \sum_{j=0}^{d} x_j w_j,$$

and

$$f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T} \boldsymbol{w} \tag{2.8}$$

implicitly assuming the we added the leading term 1 to $\boldsymbol{x}$.

Linear regression makes the fundamental assumption that the true function $\tilde{f}(\boldsymbol{x})$ is approximately linear in its arguments,

$$\tilde{f}(\boldsymbol{x}) \approx \boldsymbol{x}^\mathsf{T} \boldsymbol{w}\,.$$

In other words, linear regression approximates the optimal prediction (2.6) by (2.8),

$$\mathbb{E}_{\mathsf{y}|\mathsf{x}}[y|\boldsymbol{x}] \approx \boldsymbol{x}^\mathsf{T} \boldsymbol{w}^*\,.$$

Now, considering the loss function $\ell(\mathsf{y}, f(\mathbf{x}))$, the optimal parameter vector $\boldsymbol{w}^*$ is the one that minimizes the expected generalization loss $L(\hat{y})$,

$$\begin{aligned}\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} L(\hat{y}) \\ &= \arg\min_{\boldsymbol{w}} \mathbb{E}_{\mathbf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[\ell(\mathsf{y}, f(\mathbf{x}))] \\ &= \arg\min_{\boldsymbol{w}} \mathbb{E}_{\mathbf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[(\mathsf{y} - \mathbf{x}^\mathsf{T}\boldsymbol{w})^2] \end{aligned} \tag{2.9}$$

Solving (2.9) requires the knowledge of the joint probability distribution $p(\boldsymbol{x}, y)$, which is in general unknown. To circumvent this problem, we can use the training set to approximate the expectation (2.9) by the empirical average over the N training points as

$$\mathbb{E}_{\mathbf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[(\mathsf{y} - \mathbf{x}^\mathsf{T}\boldsymbol{w})^2] \approx \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w})^2 .$$

Thus, the optimal parameter vector $\boldsymbol{w}^*$ is obtained as

$$\begin{aligned}\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w})^2 \\ &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w})^2 . \end{aligned} \tag{2.10}$$

Finding the linear function $f(\boldsymbol{x})$ that minimizes (2.10) is referred to as *least squares regression*. The quantity $\sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w})^2$ is sometimes called the residual sum of squares (RSS), and $r_i = y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w}$ the residual for y_i. The common least squares solution approximates the conditional expectation by an empirical average over the training set.

We can rewrite (2.10) as

$$\begin{aligned}\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w})^2 \\ &= \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 \\ &= \arg\min_{\boldsymbol{w}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) . \end{aligned} \tag{2.11}$$

In the equation, $\boldsymbol{X}$ is the $N \times (d+1)$ matrix that collects all input vector of the training set (with a leading 1),

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \dots & x_{N,d} \end{pmatrix} \tag{2.12}$$

and $\boldsymbol{y} = (y_1, \dots, y_N)^\mathsf{T}$ is the length-N vector collecting the outputs in the training set.

(2.11) can be solved by differentiating the expression with respect to $\boldsymbol{w}$,

$$\begin{aligned}\frac{\partial\,(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})}{\partial \boldsymbol{w}} &= \frac{\partial\,\boldsymbol{y}^\mathsf{T}\boldsymbol{y} - 2\boldsymbol{w}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{y} + \boldsymbol{w}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{w}}{\partial \boldsymbol{w}} \\ &= -2\boldsymbol{X}^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) . \end{aligned}$$

Assuming that $\boldsymbol{X}$ has full column rank, and hence $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is non-singular, we can set the derivative to zero,

$$\boldsymbol{X}^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) = 0\,. \tag{2.13}$$

Solving for $\boldsymbol{w}$ we obtain

$$\boldsymbol{w}^* = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}\,. \tag{2.14}$$

More in general, each output y may be a vector of length K, in which case $\boldsymbol{w}$ would be a matrix of dimensions $d \times K$ coefficients.

2.2.1 Application to classification

Consider now the binary classification problem where y takes on two classes $\{a, b\}$. We can easily modify linear regression for this problem. To do so, we simply encode $y = a \Longrightarrow y = 0$ and $y = b \Longrightarrow y = 1$ and then we apply linear regression as it is and use it to predict the output y by thresholding the resulting model

$$\hat{y}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|\boldsymbol{x}] = \boldsymbol{x}^\mathsf{T}\boldsymbol{w}^* \leq 0.5 \\ 1 & \text{otherwise} \end{cases}\,. \tag{2.15}$$

In this context, the linear model determines a decision boundary which is the hyperplane $\{\boldsymbol{x} : \boldsymbol{x}^\mathsf{T}\boldsymbol{w}^* = 0.5\}$. Note that in the classification rule (2.15), we can interpret $f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|x] = \boldsymbol{x}^\mathsf{T}\boldsymbol{w}$ as the probability $p(\mathsf{y} = 1|\boldsymbol{x})$. If this probability is less than or equal to 0.5, we decide for class 0 (i.e., a). Otherwise we decide for class 1 (i.e., b).

We apply the linear model for classification to a binary classification problem with training data shown in Figure 2.1(left). The data consists of 100 samples from two bivariate Gaussian distributions of different means. The blue circles are generated according to $\mathcal{N}((1, 0)^\mathsf{T}, \boldsymbol{I})$, while the orange circles are generated according to $\mathcal{N}((0, 1)^\mathsf{T}, \boldsymbol{I})$

For a new previously unseen set of coordinates $\boldsymbol{x} = (x_1, x_2)$, we would like to determine to which of the two bivariate Gaussians the coordinates correspond to, i.e., we have a binary classification problem. We apply linear regression by encoding class orange to 1 and class blue to 0. The outcome of the thus built classifier is then by (2.15). The result is shown in Figure 2.1(right). The colored regions indicate the points in the input space that are classified as blue or orange. According to (2.15), the set of points in $\mathbb{R}^2$ classified as orange corresponds to $\{\boldsymbol{x} : \boldsymbol{x}^\mathsf{T}\boldsymbol{w} > 0.5\}$. As it can be seen from the figure, the linear model is not a good model for the classification task—the linear classier misclassifies quite a few of the training examples. The linear model is indeed too rigid: By simple inspection, we see that the two classes cannot be separated by a line.

2.3 k-Nearest neighbor regression

We now consider another simple method for regression, called k-nearest neighbor regression. Given an input $\boldsymbol{x}$, k-nearest neighbor regression predicts the outcome y as an average over the outputs y corresponding to the k nearest neighbors of $\boldsymbol{x}$ in the training data, i.e.,

$$\hat{y}(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} y_i(\boldsymbol{x}_i)\,, \tag{2.16}$$

where $\mathcal{N}_k(\boldsymbol{x})$ denotes the set of k nearest neighbors to $\boldsymbol{x}$ in the training set, i.e., the k closest points $\boldsymbol{x}_i$.

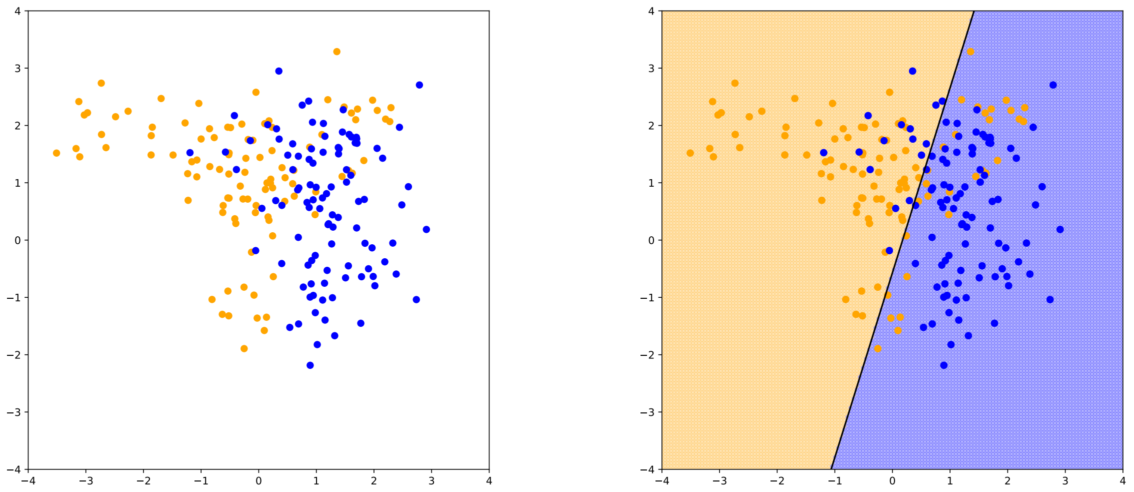

Figure 2.1: (Left)Training data set generated from two bivariate Gaussians of different mean (100 samples for each). (Right) Classification applying the linear regression model.

The key to defining nearest neighbor methods is the definition of closeness and hence neighborhood. When $\boldsymbol{x} \in \mathbb{R}^d$, the simplest approach is to use the Euclidean distance $d(\boldsymbol{x}, \boldsymbol{x}_i) = \|\boldsymbol{x} - \boldsymbol{x}_i\|^2$, and $\mathcal{N}_k(\boldsymbol{x})$ will be set of vectors $\boldsymbol{x}_i$ closest (in Euclidean distance) to $\boldsymbol{x}$. Different definitions of distance are obviously possible, and of course will affect the model quality, as well as the selection of the number of neighbors k we consider.

Consider now again optimal inference with a quadratic loss function. We recall that the optimal solution is given by (see (2.6))

$$\hat{y}(x) = f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}]\,.$$

We can interpret the k-nearest neighbors method as trying to accomplish this directly. We would like to compute the average over all $\boldsymbol{y}$ corresponding to the input $\boldsymbol{x}$. However, since there is typically at most one observation at any point $\boldsymbol{x}$, we approximate this by

1. Replacing $\mathbf{x} = \boldsymbol{x}$ by the neighborhood of $\boldsymbol{x}$ in the training data $\mathcal{N}_k(\boldsymbol{x})$.

2. Replacing the expectation by the average over the k training neighbors.

Hence, we are making two approximations,

- The expectation is approximated by the sample average;

- Conditioning at a point $\boldsymbol{x}$ is relaxed to conditioning on some region close to the target point.

We can think of changing k as balancing between the inaccuracies of the two approximations. When we increase k:

- The average is more accurate and stable (we can think of this as reducing variance), but

- The neighborhood is bigger and less representative of $\mathbf{x} = \boldsymbol{x}$ (increases the bias).

It can be shown that, under mild regularity conditions on the joint probability distribution $p(\boldsymbol{x}, y)$, as $N \to \infty$, $k(N) \to \infty$ and $k(N)/N \to 0$, $\hat{y}(\boldsymbol{x}) \to \tilde{f}(\boldsymbol{x}),\ \forall \boldsymbol{x}$.

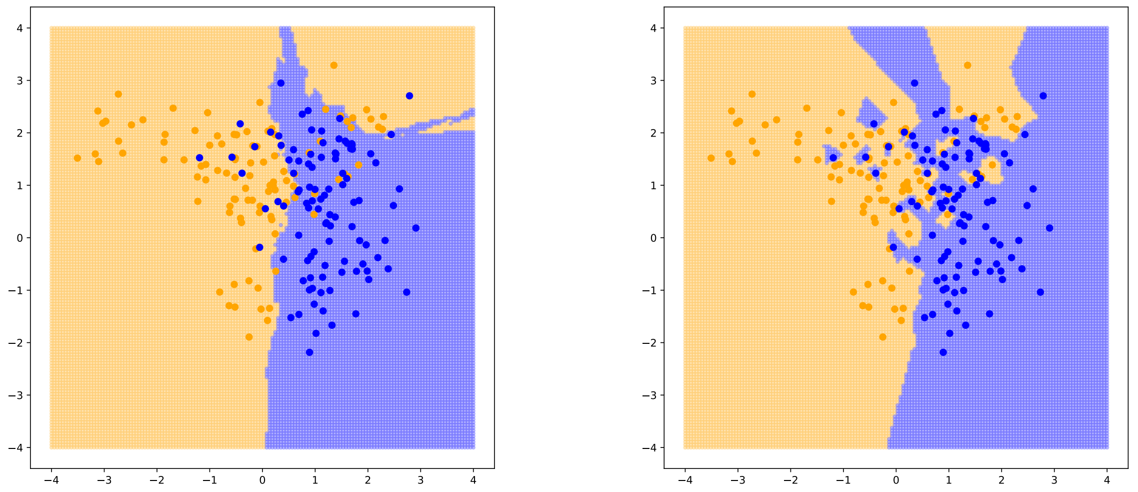
Figure 2.2: Classification applying the k-nearest neighbor method with $k = 15$ (left) and $k = 1$ (right).

2.3.1 Application to classification

The application of the k-nearest neighbor method to classification is straightforward. We can simply replace the averaging in the regression setting by a majority vote, i.e.,

$$\hat{y}(\boldsymbol{x}) = \mathbb{1}\left\{\frac{1}{k}\sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{1}\{y_i(\boldsymbol{x}_i) = 1\} > 0.5\right\}.$$

In words, the k-nearest neighbor classifier works as follows,

1. Find the k predictors $\mathcal{N}_k(\boldsymbol{x})$ in the training data set that are closest to the input $\boldsymbol{x}$.
2. Majority vote: Assign $\boldsymbol{x}$ to the class that most predictors in $\mathcal{N}_k(\boldsymbol{x})$ belong to.

We now apply the k nearest neighbor regression to the same classification problem as before. In Figure 2.2(left), we plot the result for the 15-nearest neighbor classifier. Again, the colored regions indicate all those points in input space classified as blue or orange. We see that the decision boundaries that separate the blue from the orange regions are far more irregular, and respond to local clusters where one class dominates.

Figure 2.2(right) shows the results for 1-nearest-neighbor classification—$\hat{y}$ is assigned the value y_i corresponding to the training data point $\boldsymbol{x}_i$ closest to $\boldsymbol{x}$. The decision boundary is even more irregular than before.

Note that the error on the training data decreases with decreasing value of k. For $k = 1$, the error is zero. Does this mean that choosing $k = 1$ is optimal? Not necessarily, because learning the training set does not mean that the resulting classifier will generalize well to new previously unseen data. To compare k-nearest neighbor classifiers with different values of k, we will need to evaluate their performance on an independent test set. This will be discussed in detail in Chapter 6.

2.3.2 Discussion

Let us summarize the two basic regression approaches. Both least squares and k-nearest neighbors attempt to approximate the conditional expectation by a sample average. However, they differ in the model assumptions:

- Least squares assumes $\tilde{f}(\boldsymbol{x})$ is well approximated by a globally linear function.
- k-nearest neighbors assumes $\tilde{f}(\boldsymbol{x})$ is well approximated by a locally constant function.

The linear decision boundary from least squares is, by construction, very smooth, and apparently stable to fit. Linear regression, however, makes the strong assumption that a linear decision boundary is appropriate. In the statistical learning language, we say that it has low variance and (potentially) high bias.

On the other hand, the k-nearest neighbor procedures do not appear to rely on any stringent assumptions about the underlying data, and can adapt to any shape of the data. However, any particular subregion of the decision boundary depends on a handful of input points and their particular positions, and is thus wiggly (not smooth—for small k) and unstable to fit (for small k). In the statistical learning lingo, it has high variance and low bias.

In addition to being important in themselves, linear regression and k-nearest neighbors represent two conceptual approaches that are shared among many methods and approaches that we will study during the course.

Parametric vs. nonparametric models

There are many ways to define models for learning, but the most important distinction is the following:

- *Parametric models*, which build $f(\boldsymbol{x})$ as a parametric model that applies to the whole space. In this case, in order to learn an approximation of the distribution $p(y|\boldsymbol{x})$ and predict y, we often proceed by first selecting a parametric model, also known as hypothesis class, with a fixed number of parameters and then by learning the parameters of the model to fit the training data $\mathcal{D}$. Many of modern learning techniques are model based.
- *Non-parametric models*, which don't make explicit assumptions about the form of function $f(\boldsymbol{x})$, but describe it in terms of local behavior of the training data in the region near $\boldsymbol{x}$. They seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly. Such approaches can have a major advantage over parametric approaches: by avoiding the assumption of a particular functional form for f, they have the potential to accurately fit a wider range of possible shapes for f. In this case, the number of parameters grows with the amount of training data.

Parametric models have the advantage of often being faster to use, but the disadvantage of making stronger assumptions about the nature of the data distributions. Nonparametric models are more flexible, but often computationally intractable for large datasets. Linear regression is an example of a parametric model, whereas k-nearest neighbors is an example of a non-parametric model.

2.4 Classification and statistical learning

We have seen the application of linear regression and k-nearest neighbors to the (binary) classification task. However, what is the optimal classifier in terms of minimizing the probability of misclassification? In this section, we answer this question.

Consider the classification problem where the output y is a categorical variable that takes values on a discrete finite set $\mathcal{C} = \{C_1, \ldots, C_K\}$ of cardinality $|\mathcal{C}| = K$ classes. We can take a similar approach to the one used for regression, i.e., we also assume that $\boldsymbol{x}$ and y are related via a function $\tilde{f}(\boldsymbol{x})$, called rule in classification. Hence, we want to learn a rule $f(\boldsymbol{x})$ which maps observed features $\boldsymbol{x}$ to one of the classes

$\{C_1, \dots, C_K\}$. To do so, we need to define a proper loss function and we will learn the model, i.e., the function $f(\boldsymbol{x})$, to minimize the expected prediction error under this loss function.

Now the question is, what is a suitable loss function for the classification problem? Obviously, we need a different loss function compared to that of regression to penalize classification errors.

If we consider the misclassification rate, a natural choice for the loss function is the so-called 0–1 loss function, defined as

$$\ell(y, f(\boldsymbol{x})) = \begin{cases} 0 & y = f(\boldsymbol{x}) \\ 1 & y \neq f(\boldsymbol{x}) \end{cases} ,$$

i.e., it assigns cost zero if the classifier classifies correctly, and cost one if the classifier misclassifies. The 0–1 loss function can be expressed in compact form as

$$\ell(y, f(\boldsymbol{x})) = \mathbb{1}\{y \neq f(\boldsymbol{x})\} .$$

The optimal rule $f^*(\boldsymbol{x})$ under the 0–1 loss function is given by

$$f^*(\boldsymbol{x}) = \underset{i \in [K]}{\operatorname{argmax}}\, p(\mathsf{y} = i | \boldsymbol{x}) , \tag{2.17}$$

which is called the Bayes' rule or Bayes' classifier, which chooses the most likely outcome at every point $\mathbf{x} = \boldsymbol{x}$ using the conditional (discrete) distribution $p(y|\boldsymbol{x})$. For simplicity, we used the mapping $C_i \mapsto i$ for $i \in [K]$, so that $y = i$ means that the output is classified to class C_i.

We prove (2.17) in the following. As before, the optimal rule is obtained by minimizing the expected prediction error,

$$\begin{aligned} f^*(\boldsymbol{x}) &= \arg\min_f \mathbb{E}_{\mathbf{x}, \mathsf{y} \sim p(\boldsymbol{x}, y)}[\ell(\mathsf{y}, f(\boldsymbol{x}))] \\ &= \arg\min_f \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{x}}}\left[\mathbb{E}_{\mathsf{y} \sim p_{\mathsf{y}|\mathbf{x}}}[\ell(\mathsf{y}, f(\boldsymbol{x}))]\right] \\ &= \arg\min_f \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{x}}}\left[\mathbb{E}_{\mathsf{y} \sim p_{\mathsf{y}|\mathbf{x}}}[\mathbb{1}\{\mathsf{y} \neq f(\boldsymbol{x})\}]\right] \end{aligned}$$

Again, we can focus on the minimization for a given $\boldsymbol{x}$ and hence,

$$\begin{aligned} f^*(\boldsymbol{x}) &= \arg\min_f \mathbb{E}_{\mathsf{y} \sim p_{\mathsf{y}|\mathbf{x}}}[\mathbb{1}\{y \neq f(\boldsymbol{x})\}] \\ &= \arg\min_f \sum_{i=1}^{K} \mathbb{1}\{i \neq f(\boldsymbol{x})\} p(\mathsf{y} = i|\boldsymbol{x}) \\ &= \arg\min_f \sum_{i \neq f(\boldsymbol{x})} p(\mathsf{y} = i|\boldsymbol{x}) \\ &= \arg\min_f \left[1 - p(\mathsf{y} = f(\boldsymbol{x})|\boldsymbol{x})\right] \\ &= \underset{f}{\operatorname{argmax}}\, p(\mathsf{y} = f(\boldsymbol{x})|\boldsymbol{x}) \\ &= \underset{j \in [K]}{\operatorname{argmax}}\, p(\mathsf{y} = j|\boldsymbol{x}) . \end{aligned}$$

In Figure 2.3, for the same classification problem that we considered in Figures 2.1 and 2.2, we show the solution given by the Bayes' classifier.

Now, going back to k-NN classification, we can think of it as directly approximating the solution (2.17) by means of an empirical approximation, in which we choose the Bayes decision based on the training points at

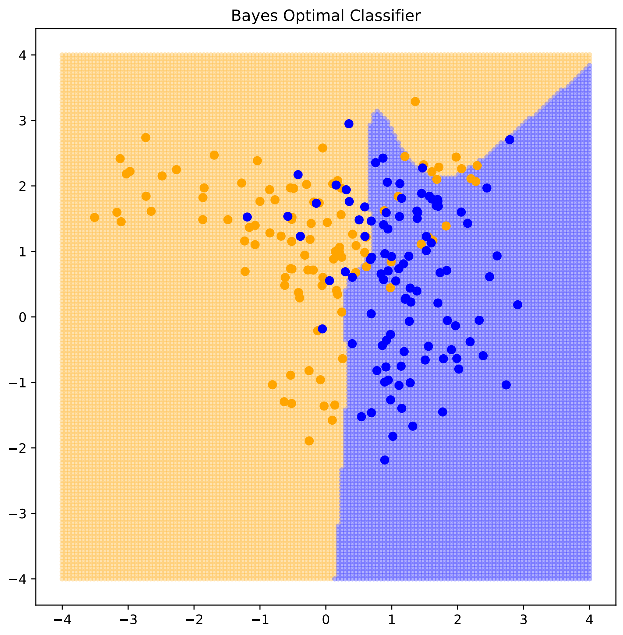

Figure 2.3: Classification applying the Bayes' classifier.

the neighborhood of $\boldsymbol{x}$: we relax the conditional probability at a point $\boldsymbol{x}$, $p(i|\boldsymbol{x})$ to the conditional probability within a neighborhood of this point, $p(i|\mathcal{N}_k(\boldsymbol{x}))$, and this probability is approximated by a sample average, i.e., with the frequency of class C_i among the k closest neighbors of $\boldsymbol{x}$. Thus, given N training data points $(y_i, \boldsymbol{x}_i)$, the optimal decision for a data point $\boldsymbol{x}$ is

$$f^*(\boldsymbol{x}) = \underset{j\in[K]}{\operatorname{argmax}} \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{1}\{y_i = j\}\,.$$

2.5 The curse of dimensionality

Nonparametric (local) methods such as k-nearest neighbors often perform poorly when the number of dimensions is large. This phenomenon is commonly referred to as the *curse of dimensionality* and we discuss it in the following.

Consider again the application of k-NNs to the regression problem. Seeing (2.16) as an approximation of (2.6), one might assume that by considering a very large training data set, we can accurately approximate the optimal solution (2.6)—we might expect that a large training set will provide a quite large number of observations close to any $\boldsymbol{x}$, and by averaging their corresponding outputs, we can well approximate y.

Consider as an example the two-dimensional data set in Figure 2.4. For this figure, the input data points $\boldsymbol{x}_i = (x_{i,1}, x_{i,2})$ (blue and red circles) are uniformly sampled from $\mathcal{X} = [-1, 1]^2$. Assume a new data point $\boldsymbol{x}$, which we locate at the coordinate $(0, 0)$ for simplicity. We would like to predict the output y corresponding to $\boldsymbol{x}$ using the k-NNs method with $k = 3$. We observe in the figure that, as the number of data points N increases, the three nearest neighbors are closer and closer to the new data point $\boldsymbol{x}$. Hence, the accuracy of the prediction will improve with increasing number of training examples. More formally, as N increases,

$$\hat{y}(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} y_i(\boldsymbol{x}_i) \longrightarrow_{N\to\infty} \mathbb{E}[\mathsf{y}|\boldsymbol{x}]$$

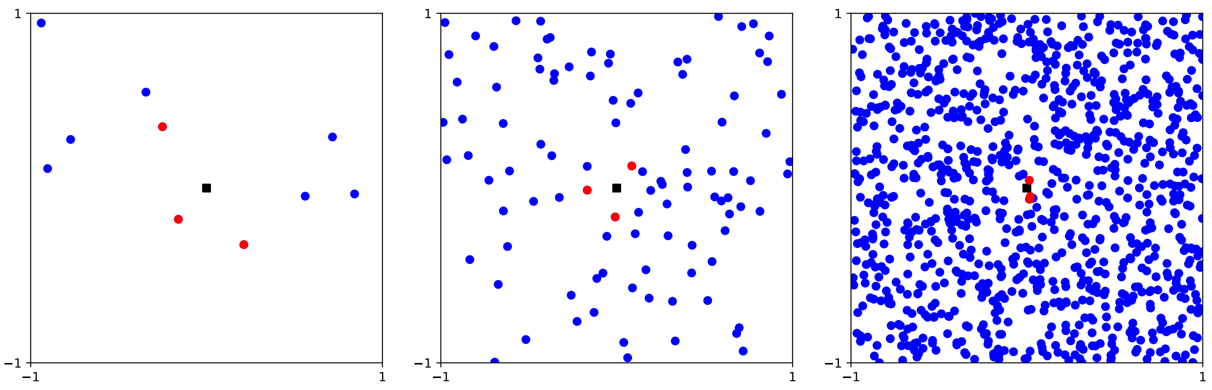

Figure 2.4: Closeness of neighbors for increasing number of training examples.

However, this intuition breaks down for a large number of dimensions. Indeed, the main problem with k-NN regression (and k-NN classifiers), as well as with other local methods, is that they do not work well with high-dimensional inputs. The poor performance in high-dimensional settings is due to the so-called *curse of dimensionality.*

The curse of dimensionality refers to the fact that high-dimensional spaces are a lot harder to fill up with points than lower-dimensional ones—we require a number of points that is exponential in the number of dimensions to fill a high-dimensional space. Hence, in order to obtain an accurate prediction, we require an amount of training data that grows exponentially with the dimensionality.

The curse of dimensionality problem manifests itself in different ways:

1. Nearest neighbors are not so close to the data point $\boldsymbol{x}$
2. The k-NNs of $\boldsymbol{x}$ are closer to the boundary of $\mathcal{X}$.
3. Need a prohibitive number of training samples to densely sample $\mathcal{X} \in \mathbb{R}^d$

Next, we discuss the first manifestation of the curse of dimensionality. For the other two, we refer the reader to [1, Sec. 2.5].

2.5.1 Problem 1: For large d, the nearest neighbors are not so close

One way of demonstrating the first phenomenon is by considering the relation between the radius of a region—which is equal to the distance to the closest neighbor—, and which portion of the space it occupies. Consider applying the k-NN procedure to data where the training inputs $\boldsymbol{x}$ are uniformly distributed in a d-dimensional unit hypercube, $\mathcal{X} = [0,1]^d$. Suppose now that we want to estimate the density of class labels around a test point $\boldsymbol{x}$ by growing a hypercube around $\boldsymbol{x}$ until it contains a desired fraction ρ of the data points. This is illustrated in Figure 2.5. What is the expected length of the side of the smallest hyper-cube containing a fraction ρ of the data points? Note that a fraction ρ of the data points corresponds to a fraction ρ of the unit volume. The volume of a d-dimensional hypercube of side r is r^d. Looking for r such that r^d equals a fraction ρ of the unit hyper-cube volume, i.e., $r^d = \rho$, we obtain that the expected side length of this cube will be $r_d(\rho) = \rho^{1/d}$.

If $p = 10$ dimensions, and we want to base our estimate on 10% of the data, i.e., want to capture a fraction $\rho = 1/10$ of the data points, we have $r_{10}(1/10) = 0.8$, so we need to grow the cube 80% along each dimension

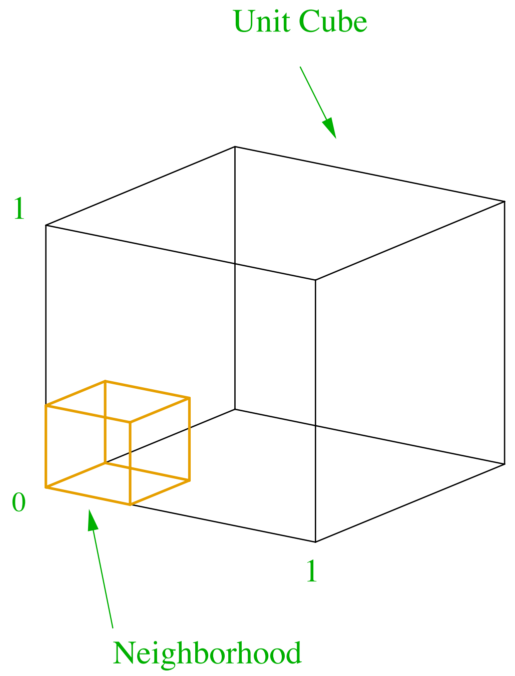

Figure 2.5: Illustration of the curse of dimensionality. The orange cube shows the neighborhood for a data point of uniform data in a unit cube. (Figure from [1])

around $\boldsymbol{x}$. If we only require to use 1% of the data to make an estimate, we obtain $r_{10}(1/100) = 0.63$. Hence, to capture 1% of the data in 10 dimensions, we already expect the closest neighbors to be very far away (the entire range of the data is only 1 along each dimension), i.e., the k-NN method is no longer very local, despite its name. Making a decision based on neighbors that are far away from the test point $\boldsymbol{x}$ may be problematic, as far-away data points may not be good predictors for the behavior of the function at the test point $\boldsymbol{x}$. Note that, for $d \to \infty$, $r \to 1$.

In conclusion, in high dimensions, the closest neighbors are very far away, unless we have an exponential amount of data in our training set. Therefore, as long as the true function $f^*(\boldsymbol{x}) = \mathbb{E}(\mathsf{y}|\mathbf{x} = \boldsymbol{x})$ is not very smooth, the nearest neighbors will not be informative for prediction. Reducing ρ so that we only consider the close neighborhood of $\boldsymbol{x}$ runs also into some problems, as we will have fewer neighbors to average upon, leading to a fit with higher variance.

2.6 Probabilistic (statistical) models for learning

The goal of supervised learning is to find an accurate approximation $f(\boldsymbol{x})$ to the true function $\tilde{f}(\boldsymbol{x})$ that underlies the predictive relationship between the inputs and outputs, and we would like to do so from labeled training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. In the learning jargon, we learn $f(\boldsymbol{x})$ from training data. The learned function will be useful if the predicted outputs for new previously unseen inputs are close enough to the real outputs. In this section, we deepen in the comprehension of the supervised learning problem. Particularly, we focus on the regression problem.

Up to now we have assumed that the relationship between the input $\boldsymbol{x}$ and the output y is deterministic. i.e., $y = f(\boldsymbol{x})$. However, in most scenarios there is not a purely deterministic relationship between the inputs and the outputs. For example, the output y may also depend on some other unmeasured variables, or there may be measurement errors. To take into consideration this uncertainty, we typically consider a *probabilistic model*. A popular probabilistic model is the model

$$y = \tilde{f}(\boldsymbol{x}) + \varepsilon, \tag{2.18}$$

which departs from the deterministic model by adding a random variable ε independent of x. ε is typically assumed to be Gaussian distributed with mean zero, such that $\mathbb{E}[\mathsf{y}|\mathsf{x} = \boldsymbol{x}] = \tilde{f}(\boldsymbol{x})$.

The additive model makes the assumption that all deviations from the deterministic relationship $y = \tilde{f}(\boldsymbol{x})$ can be captured by the additive error ε—we express our uncertainty over the value of $f(\boldsymbol{x})$ by adding some Gaussian noise ε. The additive error model is a reasonable and convenient approximation to the reality. However, obviously, the assumption that the errors are independent of the input and identically distributed does not necessarily hold in practice.

The fundamental question now is, how do we effectively use the training data $\mathcal{D}$ to guide the learning of $f(\boldsymbol{x})$ (the approximation of $\tilde{f}(\boldsymbol{x})$)? As we have mentioned previously, the space of all possible regression functions $f(\boldsymbol{x})$ is enormous. To restrict this space and make the learning more tractable, we typically consider a parametric form of $f(\boldsymbol{x})$, denoted by $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, with a set of parameters $\boldsymbol{\theta}$. Linear regression, in particular, assumes the relationship between $\boldsymbol{x}$ and y is mostly linear, i.e.,

$$\begin{aligned}\hat{y} &= \sum_{j=0}^{d} x_j w_j + \varepsilon \\ &= \boldsymbol{x}^{\mathsf{T}} \boldsymbol{w} + \varepsilon\,. \end{aligned} \tag{2.19}$$

In this case, the parameters of the model are $\boldsymbol{\theta} = \boldsymbol{w}$.

The key property of the model (2.19) is that it is a linear function of the parameters $w_0, \ldots, w_d$. Note that it is a linear function of the input variables x_i as well, which imposes significant limitations on the model.[3] To avoid this limitation, this class of models can be extended by considering linear combinations of fixed nonlinear functions of the input variables. In general, we can imagine applying any function ϕ to $\boldsymbol{x}$ to serve as the inputs to our linear regression model, such that we obtain

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{M} \theta_i \phi_i(\boldsymbol{x})$$

where ϕ_j are known as basis functions or basis expansion and $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called a linear basis expansion. The resulting model is much richer (it is a nonlinear function of the data $\boldsymbol{x}$), but since ϕ is a fixed transformation, it is a still linear function in $\boldsymbol{\theta}$. Functions of this form are called linear models, because they are linear in $\boldsymbol{\theta}$. We will discuss such models in Chapter 5.

2.7 Maximum likelihood learning

Assume now that the model $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is fixed, and that we are interested in learning the model parameters $\boldsymbol{\theta}$. We could of course estimate the parameters $\boldsymbol{\theta}$ as we did before for the linear model by minimizing the residual sum-of-squares (see (2.10))

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \tag{2.20}$$

[3]Linear models have significant limitations for several scenarios, particularly for problems involving input spaces of high dimensionality. However, they have nice analytical properties and form the foundation for more sophisticated learning models.

For the linear model we obtained a simple closed-form solution to the minimization problem (see Section 2.2). This is also true for the basis function methods, if the basis functions do not have any hidden parameters. Otherwise, the solution requires either iterative methods or numerical optimization.

We can consider other criteria to find $\boldsymbol{\theta}$ rather than minimizing the RSS. A reasonable criterion is the maximum likelihood (ML) criterion. The ML criterion selects a value of $\boldsymbol{\theta}$ under which the training set $\mathcal{D}$ has the maximum probability of being observed. In other words, the value $\boldsymbol{\theta}$ selected by ML is the most likely to have generated the observed training set. Note that there might be more than one such value.

Let $\boldsymbol{x}_{\mathcal{D}} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ the input data and $y_{\mathcal{D}} = \{y_1, \ldots, y_N\}$ the output data. To proceed, we need to write the probability (density) of the observed outputs $y_{\mathcal{D}}$ in the training set $\mathcal{D}$ given the corresponding input data points $\boldsymbol{x}_{\mathcal{D}}$. Assuming that the data points are independent, this probability is given by

$$p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\,.$$

Taking the logarithm of this expression yields the log-likelihood (LL) function

$$\ln p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) = \sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\,.$$

The ML learning problem corresponds to choose $\boldsymbol{\theta}$ that maximizes the a posteriori probability $p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta})$ or, equivalently, the log-likelihood function,

$$\begin{aligned}
\boldsymbol{\theta}_{\mathsf{ML}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \ln p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} -\ln p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} -\sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} -\frac{1}{N}\sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\,. \qquad (2.21)
\end{aligned}$$

This criterion (2.21) is also referred to as cross-entropy loss.

For $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, it follows that $\mathsf{y}_i|\mathbf{x} \sim \mathcal{N}(y_i; f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \sigma^2)$, i.e.,

$$p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2}{2\sigma^2}}\,.$$

We get

$$\begin{aligned}
\sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) &= \sum_{i=1}^{N} \ln \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2}{2\sigma^2}} \\
&= -\frac{1}{2}\sum_{i=1}^{N} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \\
&= -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2\,.
\end{aligned}$$

Using this in (2.21) we then obtain

$$\begin{aligned}
\boldsymbol{\theta}_{\mathsf{ML}} &= \arg\min_{\boldsymbol{\theta}} -\frac{1}{N}\sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \frac{1}{2}\ln(2\pi\sigma^2) + \frac{1}{2\sigma^2 N}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \\
&= \arg\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \,,
\end{aligned}$$

which coincides with minimizing the RSS and subsequently with the least squares regression criterion (2.20) when $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}$.

2.8 The need of structured regression models

A way to think about fitting a predictive model is the following: We have a large class of candidate model functions, $\mathcal{F}$, and we want to choose a good function $f \in \mathcal{F}$ based on the training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$, $i \in [N]$, typically by minimizing a given loss function $L(\hat{y}) = L(f; \mathcal{D})$.

Consider the least squares regression criterion, which minimizes the RSS,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \ \mathsf{RSS} \triangleq \sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \,.$$

If $\mathcal{F}$ is the set of all possible functions, there are infinitely many functions $f \in \mathcal{F}$ that make $\mathsf{RSS}(f(\boldsymbol{x})) = 0$—any function $f(\boldsymbol{x})$ that passes through the training points $(\boldsymbol{x}_i, y_i)$.

The crucial point is that not all these functions will be equally good at predicting the output of new previously unseen inputs, i.e., some won't generalize well to new data. Thus, we shouldn't consider arbitrary functions, but a smaller set of functions $\mathcal{F}' \subset \mathcal{F}$ by imposing some constraints on the function f. One way of imposing some constraints is to restrict ourselves to parametric functions $f_{\boldsymbol{\theta}}$. But now the question is: which other constraints should we consider?

Least squares regression follows this recipe exactly, with $\mathcal{F}$ being the family of all linear functions, and the loss function L the RSS on the training data $\mathcal{D}$. More in general, the constraints imposed by most learning methods can be seen as restricting the complexity of the function f. Intuitively, for example, we may prefer smoother functions in $\mathcal{F}$ over others, a property which, informally, we can state as follows: the outputs corresponding to input points that are close to each other should also be close to each other.

The techniques used to restrict the regression or classification candidate model functions loosely fall into a number of classes. We consider three of them in the following.

2.8.1 Class 1: Roughness penalty

Assume that we have a measure of "niceness" (e.g., smoothness) $J(f)$. We can write the resulting problem as a constrained optimization problem,

$$f(\boldsymbol{x}) = \arg\min_{f\in\mathcal{F}:L(f;\mathcal{D})=0} J(f)\,. \tag{2.22}$$

Hence, we are looking for the function that interpolates the data with the minimum lack of smoothness.

A concrete example of this recipe is *smoothing splines interpolation* models, which we discuss in Chapter 5. Assume one-dimensional data $x \in [0,1]$. In this case, $\mathcal{F}$ is the family of all twice-differentiable functions, and we choose $J(f)$ to be a natural smoothness criterion,

$$J(f) = \int_0^1 (f''(x))^2 \mathrm{d}x\,. \tag{2.23}$$

More generally, we can relax the requirement that $L(f;\mathcal{D}) = 0$ with a trade-off between loss and smoothness, and penalize the RSS with a roughness penalty,

$$f(\boldsymbol{x}) = \arg\min_{f \in \mathcal{F}} \; L(f;\mathcal{D}) + \lambda J(f)\,. \tag{2.24}$$

The function $L(f;\mathcal{D})$ ensures that f predicts well the training values, while the functional $J(f)$ penalizes the roughness of f—it measures the smoothness of f; $J(f)$ will be large for functions f that vary too rapidly over small regions of input space, i.e., it will penalize non-smooth functions. The amount of penalty is dictated by parameter λ.

The penalty function (2.23) is a very popular one for one-dimensional data. For wiggly functions $f(x)$ this functional will have a large value, while for linear $f(x)$ it is zero. Note that for $\lambda = 0$ no penalty is imposed, and any interpolating function will do, while for $\lambda = \infty$, only functions linear in $\boldsymbol{x}$ are permitted. Obviously, other penalty functions are possible.

Models with a penalty function are referred to as *regularization* methods. The regularization expresses our prior belief that the type of functions we seek exhibit a certain type of smooth behavior.

2.8.2 Class 2: Kernel methods

Kernel methods estimate the regression or classifcation function in a local neighborhood. We need to specify the nature of the local neighborhood and the class of functions used for the local fit.

The simplest form of kernel estimate is the Nadaraya-Watson weighted average

$$f(\boldsymbol{x}) = \frac{\sum_{i=1}^N K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i) y_i}{\sum_{i=1}^N K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)}$$

where $K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)$, which is called the Kernel function, assigns weights to $\boldsymbol{x}_i$ depending on its closeness to $\boldsymbol{x}$.

Note that the k-nearest neighbor method can be seen as a kernel method with

$$K_k(\boldsymbol{x}, \boldsymbol{x}_i) = \mathbb{1}\{\|\boldsymbol{x}_i - \boldsymbol{x}\| \leq \|\boldsymbol{x}_{(k)} - \boldsymbol{x}\|\}\,,$$

where $\boldsymbol{x}_{(k)}$ is the k-th closest input in the data set to $\boldsymbol{x}$.

A popular kernel is the Gaussian kernel,

$$K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i) = \frac{1}{\lambda} \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{2\lambda} \right)$$

which assigns weights to training data points $\boldsymbol{x}_i$ that decrease exponentially with their squared Euclidean distance to the test point $\boldsymbol{x}$. Here, parameter λ is typically known as the bandwidth of the kernel.

More in general, we define a local regression estimate of $\tilde{f}(\boldsymbol{x})$ from the training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$ as $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, where $\boldsymbol{\theta}$ is chosen to minimize

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2$$

for some parametrized base regression function $f_{\boldsymbol{\theta}}$.

2.8.3 Class 3: Basis functions and dictionary methods

In this case, f is modeled as a linear expansion of basis functions,

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{M} \theta_i \phi_i(\boldsymbol{x}) \,,$$

which we have introduced earlier and sometimes is refer to as a linear model because it is linear with respect to $\boldsymbol{\theta}$. We will consider the use of basis functions in Chapter 5.

2.9 The bias–variance trade-off

The models we have encountered so far, as well as those we will see in subsequent chapters, comprise one or several parameters that control their capacity to fit the training data. For the k-NNs method, the parameter that controls the model's capacity is the number of nearest neighbors k; for linear models, this parameter is the number of basis functions M; and for roughness penalty methods, the weight of the penalty term, λ.

The smaller k in k-NNs and the larger M in linear models with M basis functions, the more capable the model is to fit the training data. In the case of the smoothing spline, the parameter λ determines models ranging from a linear fit to a model that perfectly interpolates the data.

The higher the capacity of the model (the higher its complexity, i.e., the higher the number degrees of freedom of the model), the more accurately it will able to fit the training data. However, learning entails the capability to predict the output y for an unseen data point $\boldsymbol{x}$. Models that perfectly learn the training data set will unlikely be able to predict well the output corresponding to new previously unseen data. On the other hand, low complexity models may not be able to capture all the variations present on the data and may also generalize poorly. So the question is, for a given model class, how should we set its complexity? And, how does increasing or decreasing the complexity of the model affect its predictive behavior?

To shed some light into these questions, it is instructive to consider the so-called *bias-variance trade-off*. The idea is that of decomposing the expected prediction error of a predictive modeling approach into its components, which we call bias and variance for the squared error loss, and more generally approximation and estimation error for the general case.

Consider again the regression model, assuming the probabilistic model (introduced in (2.18))

$$y = \tilde{f}(\boldsymbol{x}) + \varepsilon, \tag{2.25}$$

with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, and consider for simplicity that $\mathsf{x} = \boldsymbol{x}$ is fixed. Note that for fixed $\boldsymbol{x}$, $\mathbb{E}[\mathsf{y}|\boldsymbol{x}] = \tilde{f}(\boldsymbol{x})$ (otherwise $\tilde{f}(\boldsymbol{x})$ is random as well). Recall that y is the true value and $f(\boldsymbol{x})$ its estimate.

We described predictive modeling as a process that inputs random training data $\mathcal{D}$, runs it through a black-box modeling approach, and outputs a model $f(\cdot)$. We then get a new prediction point $\boldsymbol{x}$, and we are interested in the prediction error $L(y, f(\boldsymbol{x}))$, where $\hat{y} = f(\boldsymbol{x})$ is our prediction for $\boldsymbol{x}$ and y is the true value.

The learning process involves making an estimate of $\tilde{f}(\boldsymbol{x})$ based on the data set $\mathcal{D}$. To interpret the uncertainty of this estimate, we can consider the following. Assume the square loss function $\ell(\mathsf{y}, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$ to assess the performance of the estimate, and suppose that we had a large number of data sets each of size N and each drawn independently from the distribution $p(y, \boldsymbol{x})$. For any given data set $\mathcal{D}$, we can run our learning algorithm and obtain a prediction function $f(\boldsymbol{x}; \mathcal{D})$. Obviously, different data sets will result in different functions and consequently in different values of the RSS. The performance of a particular learning algorithm can then assessed by taking the average over this ensemble of data sets. In particular, considering all randomness, the quantity of interest is

$$\mathsf{Err}(y, f(\boldsymbol{x})) = \mathbb{E}_{\mathcal{D},\mathsf{x},\mathsf{y}}[\ell(\mathsf{y}, f(\mathbf{x}))]\,,$$

which is referred to as the expected prediction errror (or expected generalization error). For a fixed input $\boldsymbol{x}$, the generalization error is

$$\begin{aligned}
\mathsf{Err}(\boldsymbol{x}) &= \mathbb{E}_{\mathsf{y}|\mathsf{x},\mathcal{D}}[\ell(\mathsf{y}, f(\boldsymbol{x}))|\mathbf{x} = \boldsymbol{x}] \\
&= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\mathbb{E}_{\mathcal{D}}[\ell(\mathsf{y}, f(\boldsymbol{x}))|\mathbf{x} = \boldsymbol{x}] \\
&= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\mathbb{E}_{\mathcal{D}}\left[(\mathsf{y} - f(\boldsymbol{x}))^2|\boldsymbol{x}\right] \\
&= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\mathbb{E}_{\mathcal{D}}\left[(\mathsf{y} - \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] + \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] + \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\left[(\mathsf{y} - \mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}])^2\right] + (\mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \mathbb{E}_{\mathsf{y}|\mathsf{x}}\left[(\mathsf{y} - \tilde{f}(\boldsymbol{x}))^2\right] + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \mathrm{Var}[\mathsf{y}|\boldsymbol{x}] + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right]. \qquad (2.26)
\end{aligned}$$

The first term is just the conditional variance of y given $\mathbf{x} = \boldsymbol{x}$, which is σ^2 (it follows from the probabilistic model (2.25)). This is an irreducible error that does not depend on the modeling approach or the training set $\mathcal{D}$, i.e., it is out of our control even if we know $\tilde{f}(\boldsymbol{x})$.

The term $\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})]$ is called the bias (and subsequently $(\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2$ the squared bias) or approximation error, and represents the extent to which the average prediction over all data sets differs from the true function (true mean) $\tilde{f}(\boldsymbol{x})$.

The third term, called the variance or estimation error, is the variance of the predicted value around its expectation over $\mathcal{D}$—it measures the extent to which the solutions for individual data sets vary around their average, and hence it measures the extent to which the function $f(\boldsymbol{x};\mathcal{D})$ is sensitive to the particular choice of data set. We rewrite (2.26) as

$$\begin{aligned}
\mathsf{Err}(\boldsymbol{x}) &= \sigma^2 + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \sigma^2 + (\underbrace{\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})]}_{\text{bias}})^2 + \mathrm{Var}_{\mathcal{D}}[f(\boldsymbol{x})]\,. \qquad (2.27)
\end{aligned}$$

This is decomposition is called the bias-variance decomposition of the squared prediction error. In addition to decomposing nicely from a mathematical perspective, it also provides us terms that clarify the sources of the prediction error:

- An irreducible error that cannot be avoided due to data randomness
- An approximation error (the bias) that is due to the limited flexibility of our modeling approach—which cannot properly express $\mathbb{E}(\mathsf{y}|\boldsymbol{x})$ even on average. The bias is caused by the simplifying (and perhaps erroneous) assumptions made by a model to make the learning easier. For example, if we assume that the relationship between $\boldsymbol{x}$ and y is linear but in reality is highly nonlinear, this will induce a high bias.
- An estimation error (variance) that measures the sensitivity/variability of the modeling approach due to the randomness in $\mathcal{D}$. The variance quantifies the amount that the estimate of the target function will change if different training data was used.

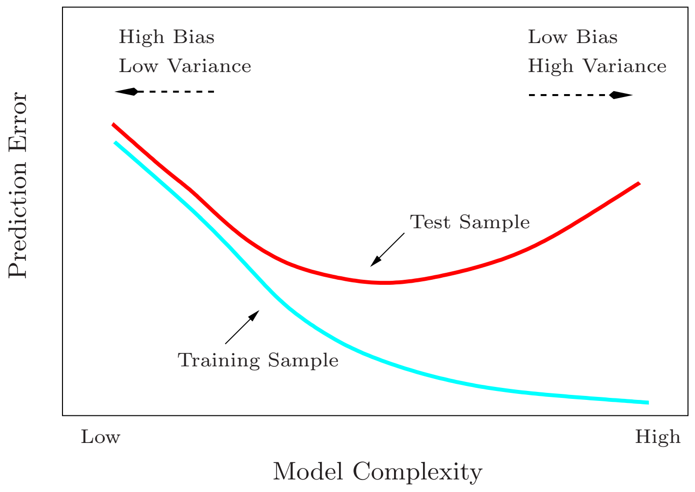

Figure 2.6: Test and training error as a function of the model complexity. (Figure from [1])

Remark 2.1 A low bias suggests less assumptions about the form of the target function, while a high bias suggests more assumptions about the form of the target function.

Remark 2.2 For loss functions other than the squared loss for prediction, we do not have this elegant mathematical decomposition, but the notions of approximation error and estimation error as underlying the performance is still important.

Our goal is to minimize the expected loss. Perhaps not surprisingly, there is a trade-off between a model's ability to minimize bias and variance—very flexible (high capacity, high complexity) models are characterized by a low bias and high variance, while relatively rigid models yield high bias and low variance. The model with the optimal predictive capability is the one that leads to the best balance between bias and variance, thus minimizing the expected prediction (or generalization) error (2.27). How should we find this model? Recall that our learning is based on the available training data $\mathcal{D}$. Should we choose the model complexity that produces the predictor which minimizes the training error? Non-surprisingly, this is not a good idea. Figure 2.6 shows the typical behavior of prediction error on the training set and on a test set (not used for training), as a function of the model complexity.

The training error decreases as we increase the model complexity, since we are able to fit the training data better. However, for high model complexity the model adapts itself too closely to the training data. We say that the model *overfits* the data: the model is too rich and tries to capture every minor variation in the training set, which is more likely to be noise than true signal. With overfitting, the model memorizes the training set, rather than learn how to generalize to unseen examples, and hence will lose the ability to generalize well (i.e., it will have a large test error). In contrast, if the model is not complex enough, it will underfit the data: the model is not rich enough to capture the underlying patterns of the data. In this case, the model may have large bias, again resulting in poor generalization.

Understanding the bias-variance trade-off is critical for understanding the behavior of prediction models. A model with high bias pays little attention to the training data and oversimplifies (underfits) the model. It hence always leads to high error on both the training and test data. In contrast, a model with high variance pays a lot of attention to the training data and does not generalize on the data which it hasn't seen before. As a result, it performs very well on training data but poorly on test data.

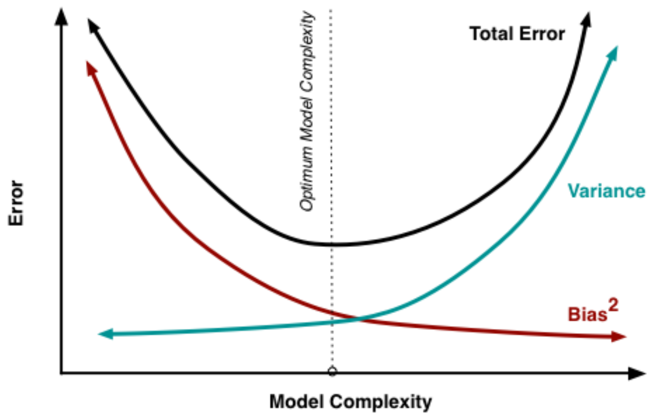

Figure 2.7: Bias and viariance contributing to the prediction error.

At the end of the day, however, what we really care about is the prediction error, not the specific decomposition. The sweet spot for any model is the level of complexity at which the increase in bias is equivalent to the reduction in variance, see Figure 2.7. If our model complexity exceeds this sweet spot, we are in effect overfitting the data, while if our complexity falls short of the sweet spot, we are underfitting.

It is important to remark that, as the number of data points increases, overfitting is avoided even for large model complexity.

We will discuss model selection in Chapter 6.

2.9.1 The bias-variance decomposition for least squares regression

Let's particularize the bias-variance decomposition in (2.27) for the least squares regression, assuming that the linear model is true, i.e., $y = \boldsymbol{x}^\top \boldsymbol{w} + \varepsilon$. For the least squares, $\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ and hence

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{x}^\top \boldsymbol{w}^* \\ &= \boldsymbol{x}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \\ &= \boldsymbol{x}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{w} + \varepsilon) \\ &= \boldsymbol{x}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w} + \boldsymbol{x}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \varepsilon \\ &= \boldsymbol{x}^\top \boldsymbol{w} + \boldsymbol{x}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \varepsilon \,, \end{aligned}$$

Therefore, as $\mathbb{E}[\varepsilon] = 0$, $\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] = \boldsymbol{x}^\top \boldsymbol{w} = \tilde{f}(\boldsymbol{x})$, and as a result $\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] = 0$, i.e., there is no bias.

Consider now the term $\mathrm{Var}_{\mathcal{D}}[f(\boldsymbol{x})]$,

$$\begin{aligned} \mathrm{Var}_{\mathcal{D}}[f(\boldsymbol{x})] &= \mathrm{Var}_{\mathcal{D}}[\boldsymbol{x}^\top \boldsymbol{w} + \boldsymbol{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \varepsilon] \\ &= \mathrm{Var}_{\mathcal{D}}[\boldsymbol{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \varepsilon] \,, \end{aligned} \tag{2.28}$$

where $\mathbf{X}$ is now random as we take an average over training sets (recall that $\boldsymbol{x}$ is fixed).

To proceed, we use the law of total variance,

$$\mathrm{Var}_{\mathsf{a}}[g(\mathsf{a})] = \mathbb{E}_{\mathsf{a}}[\mathrm{Var}_{\mathsf{a}}[g(\mathsf{a})|\mathsf{a}]] + \mathrm{Var}_{\mathsf{a}}[\mathbb{E}_{\mathsf{a}}[g(\mathsf{a})|\mathsf{a}]] \,,$$

so that we can write (2.28) as

$$\mathrm{Var}_{\mathcal{D}}[f(\boldsymbol{x})] = \mathbb{E}_{\mathcal{D}}\left[\mathrm{Var}_{\mathcal{D}}\left[\boldsymbol{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \varepsilon | \mathbf{X}\right]\right] + \mathrm{Var}_{\mathcal{D}}\left[\mathbb{E}_{\mathcal{D}}\left[\boldsymbol{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \varepsilon | \mathbf{X}\right]\right]$$

Note that

$$\mathbb{E}_{\mathcal{D}}\Big[\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\boldsymbol{\varepsilon}|\mathbf{X}\Big] = 0$$

as $\mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}$, and thus the second term is zero.

The first term is

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}\Big[\mathrm{Var}_{\mathcal{D}}\Big[\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\boldsymbol{\varepsilon}|\mathbf{X}\Big]\Big] &= \mathbb{E}_{\mathcal{D}}\Big[\Big(\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\Big)\mathsf{Cov}[\varepsilon]\Big(\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\Big)^{\mathsf{T}}\Big] \\
&= \sigma^2\mathbb{E}_{\mathcal{D}}\Big[\Big(\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\Big)\Big(\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\Big)^{\mathsf{T}}\Big] \\
&= \sigma^2\mathbb{E}_{\mathcal{D}}\Big[\Big(\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\Big)\boldsymbol{X}\Big((\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\Big)^{\mathsf{T}}\boldsymbol{x}\Big] \\
&= \sigma^2\mathbb{E}_{\mathcal{D}}\Big[\Big(\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\Big)\boldsymbol{X}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\boldsymbol{x}\Big] \\
&= \sigma^2\mathbb{E}_{\mathcal{D}}\Big[\boldsymbol{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\boldsymbol{x}\Big] \\
&= \sigma^2\boldsymbol{x}^{\mathsf{T}}\mathbb{E}_{\mathcal{D}}\Big[(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\Big]\boldsymbol{x}
\end{aligned}$$

With this, for least-squares regression we obtain

$$\mathsf{Err}(\boldsymbol{x}) = \sigma^2 + \sigma^2\boldsymbol{x}^{\mathsf{T}}\mathbb{E}_{\mathcal{D}}\Big[(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\Big]\boldsymbol{x}\,. \tag{2.29}$$

Now let's consider the expected generalization loss $\mathsf{Err}(y, f(\boldsymbol{x}))$, i.e., we need to average over $\boldsymbol{x}$,

$$\begin{aligned}
\mathsf{Err}(y, f(\boldsymbol{x})) &= \mathbb{E}_{\mathsf{x}}[\mathsf{Err}(\mathsf{x})] \\
&= \sigma^2 + \sigma^2\mathbb{E}_{\mathsf{x},\mathcal{D}}\Big[\mathsf{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathsf{x}\Big]\,.
\end{aligned}$$

Assuming large N and $\mathbb{E}[\mathsf{x}] = 0$, then $\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X} \to N\mathsf{Cov}[\mathsf{x}']$, therefore

$$\mathsf{Err}(y, f(\boldsymbol{x})) = \sigma^2 + \sigma^2\mathbb{E}_{\mathsf{x}}\left[\mathsf{x}^{\mathsf{T}}(N\mathsf{Cov}[\mathsf{x}])^{-1}\mathsf{x}\right]\,.$$

Here the trick is to write this expression as an expectation of a trace of a 1×1 matrix and then use the cyclic property of the trace operator $\mathrm{trace}(\boldsymbol{AB}) = \mathrm{trace}(\boldsymbol{BA})$,

$$\begin{aligned}
\mathsf{Err}(y, f(\boldsymbol{x})) &= \sigma^2 + \sigma^2\mathbb{E}_{\mathsf{x}}\left[\mathsf{x}^{\mathsf{T}}(N\mathsf{Cov}[\mathsf{x}'])^{-1}\mathsf{x}\right] \\
&= \sigma^2 + \frac{\sigma^2}{N}\mathbb{E}_{\mathsf{x}}\left[\mathrm{trace}\big(\mathsf{x}^{\mathsf{T}}(\mathsf{Cov}[\mathsf{x}'])^{-1}\mathsf{x}\big)\right] \\
&= \sigma^2 + \frac{\sigma^2}{N}\mathbb{E}_{\mathsf{x}}\left[\mathrm{trace}\big((\mathsf{Cov}[\mathsf{x}'])^{-1}\mathsf{x}\mathsf{x}^{\mathsf{T}}\big)\right] \\
&= \sigma^2 + \frac{\sigma^2}{N}\mathrm{trace}\big(\mathbb{E}_{\mathsf{x}}\left[(\mathsf{Cov}[\mathsf{x}'])^{-1}\mathsf{x}\mathsf{x}^{\mathsf{T}}\right]\big) \\
&= \sigma^2 + \frac{\sigma^2}{N}\mathrm{trace}\big((\mathsf{Cov}[\mathsf{x}'])^{-1}\mathbb{E}_{\mathsf{x}}\left[\mathsf{x}\mathsf{x}^{\mathsf{T}}\right]\big) \\
&= \sigma^2 + \frac{\sigma^2}{N}\mathrm{trace}\big((\mathsf{Cov}[\mathsf{x}'])^{-1}\mathsf{Cov}[\mathsf{x}]\big) \\
&= \sigma^2 + \frac{\sigma^2}{N}\mathrm{trace}(\boldsymbol{I}_d) \\
&= \sigma^2 + \sigma^2\frac{p}{N}\,.
\end{aligned}$$

We see that the expected generalization error increases linearly as a function of d, with slope σ^2/N. However, if N is large enough and/or σ^2 is small, this growth is negligible.

Chapter 3

Linear methods for regression

In this chapter, we will delve deeper into linear regression. We recall that a linear regression model assumes that the true regression function $\tilde{f}(\boldsymbol{x}) = \mathbb{E}(\mathsf{y}|\boldsymbol{x})$ well approximated by a linear function of its inputs $x_1, \ldots, x_d$. Linear regression imposes a very strong assumption on the data and may not perform well in some scenarios so why should we even consider this simple model? Interestingly, the linear model

- Is simple and interpretable, providing a description of how the inputs affect the output;
- Can outperform non-linear methods when one has a small number of training samples, very noisy data, or sparse data;
- Can be used to model nonlinear relationships by applying a nonlinear transformation to the data via basis functions $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_M(\boldsymbol{x}))$ and replacing $\boldsymbol{x}$ by $(\phi_1(\boldsymbol{x}), \ldots, \phi_M(\boldsymbol{x}))$, as we briefly discussed in the previous chapter and will discuss in more detail in Chapter 5. The result is that the model is still linear in the parameters;
- Important for understanding nonlinear methods; many nonlinear techniques are direct generalizations of linear methods.

3.1 Linear regression models and least squares

As usual, we will assume that the data arises from the probabilistic model

$$y = \tilde{f}(\boldsymbol{x}) + \varepsilon \,,$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. We model the true, unknown function $\tilde{f}(\boldsymbol{x})$ as

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} x_j w_j \,, \tag{3.1}$$

where w_0 is the intercept and $\boldsymbol{x}^\mathsf{T} = (x_1, \ldots, x_d)$, $x_i \in \mathbb{R}^d$. We will usually write

$$f(\boldsymbol{x}) = \sum_{j=0}^{p} x_j w_j = \boldsymbol{x}^\mathsf{T}\boldsymbol{w},$$

where $\boldsymbol{w} = (w_0, w_1, \ldots, w_d)^\mathsf{T}$ and $\boldsymbol{x} = (1, x_1, \ldots, x_d)^\mathsf{T}$.

Here the w's are unknown parameters that we want to determine, and the input $\boldsymbol{x}$ is the vector of variables, which can be quantitative inputs or the result of some transformation $\boldsymbol{x} = \boldsymbol{\phi}(\tilde{\boldsymbol{x}})$ on some vector of quantitative inputs $\tilde{\boldsymbol{x}}$, so that the model is nonlinear in $\tilde{\boldsymbol{x}}$. Regardless the nature of $\boldsymbol{x}$, the model is linear in the parameters $\boldsymbol{w} = (w_0, w_1, \dots, w_d)^\mathsf{T}$.

Assuming a data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$ consisting of N training examples, as we have seen, the parameters $(w_0, w_1, \dots, w_d)$ can be optimized by, for example, minimizing the RSS

$$\begin{aligned}\mathsf{RSS}(\boldsymbol{w}) &= \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i))^2 \\ &= \sum_{i=1}^{N} \left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j} w_j \right)^2 \\ &= \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w} \right)^2 , \end{aligned} \tag{3.2}$$

where $x_{i,j}$ is the j-th entry of vector $\boldsymbol{x}_i$.

Let $\boldsymbol{X}$ be the $N \times (d+1)$ matrix that collects all input vectors of the training set (see (2.12)) and $\boldsymbol{y}$ the $N \times 1$ vector that collects the corresponding outputs. The solution of the minimization problem is (see (2.14))

$$\boldsymbol{w}^* = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y} \tag{3.3}$$

if $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is non-singular, which is obtained by differentiating the RSS with respect to $\boldsymbol{w}$ and setting the derivative to zero, i.e., solving (see (2.13))

$$\boldsymbol{X}^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) = 0 \tag{3.4}$$

for $\boldsymbol{w}$.

The predicted value $\hat{y}$ corresponding to an input vector $\boldsymbol{x} = (1, x_1, \dots, x_d)$ is then obtained as $\hat{y} = \boldsymbol{x}^\mathsf{T}\boldsymbol{w}^*$, and the fitted values to the training inputs are

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^* = \boldsymbol{X}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y} . \tag{3.5}$$

3.1.1 Geometric interpretation of the least squares

At this point, it is instructive to consider an elegant geometrical interpretation of the least-squares solution (3.3). If $d < N$, i.e., we have more samples than features, the columns of $\boldsymbol{X}$ span a linear subspace $\mathcal{S}$ of dimensionality $d+1$ which is embedded in N dimensions—a $(d+1)$-dimensional subspace of $\mathbb{R}^N$, also referred to as column space of $\boldsymbol{X}$. Let $\boldsymbol{x}^{(i)}$ denote the i-th column of $\boldsymbol{X}$, which is a vector in $\mathbb{R}^N$. (This should not be confused with $\boldsymbol{x}_i \in \mathbb{R}^d$, which represents the i-th data vector). Similarly, $\boldsymbol{y} = (y_1, \dots, y_N)^\mathsf{T}$ is a vector in $\mathbb{R}^N$. Also, note that the vector of estimated outputs, $\hat{\boldsymbol{y}}$, is $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^*$, with $\hat{y}_i = \boldsymbol{x}_i^\mathsf{T}\boldsymbol{w}^*$.

Note that $\hat{\boldsymbol{y}}$ is an arbitrary linear combination of the vectors $\{\boldsymbol{x}^{(i)}\}$, $\hat{\boldsymbol{y}} \in \operatorname{span}(\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(d+1)}\})$, hence it can live anywhere in the $(d+1)$-dimensional subspace spanned by $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(d+1)}\}$. We can write it as

$$\hat{\boldsymbol{y}} = w_0^* \boldsymbol{x}^{(1)} + \dots w_d^* \boldsymbol{x}^{(d+1)} = \boldsymbol{X}\boldsymbol{w}^* .$$

We therefore seek a vector $\hat{\boldsymbol{y}} \in \mathbb{R}^N$ that lies in the $(d+1)$-dimensional linear subspace defined by the columns of $\boldsymbol{X}$ and is as close as possible to $\boldsymbol{y}$, i.e., we want to find

$$\arg \min_{\hat{\boldsymbol{y}} \in \operatorname{span}(\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(d+1)}\})} \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 .$$

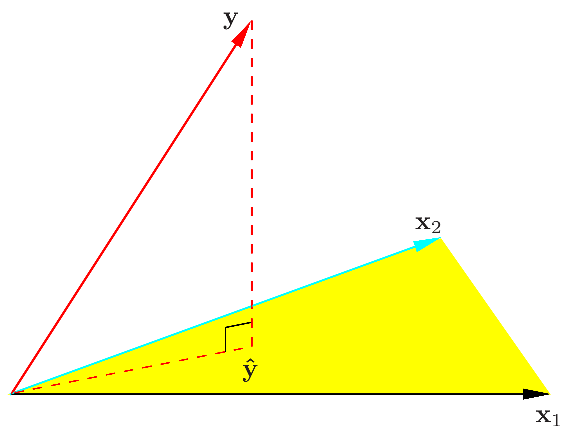

Figure 3.1: Geometrical interpretation of the least-squares solution, in an N-dimensional space considering a single feature $p = 1$. The least-squares regression function is obtained by finding the orthogonal projection of the outcome vector $\boldsymbol{y}$ onto the hyperplane spanned by the vectors $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$. The projection $\hat{\boldsymbol{y}}$ represents the vector of the least squares predictions. (Figure from [1])

To minimize the norm of the residual, $\boldsymbol{y} - \hat{\boldsymbol{y}}$, we choose $\boldsymbol{w}$ such that $\boldsymbol{y} - \hat{\boldsymbol{y}}$ is orthogonal to the subspace spanned by the column vectors of $\boldsymbol{X}$, i.e., we want the residual vector to be orthogonal to every column of $\boldsymbol{X}$. This orthogonality is expressed by (3.4) and the resulting estimate $\hat{\boldsymbol{y}}$ is hence the orthogonal projection of $\boldsymbol{y}$ onto this subspace. The projected value is given by

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^* = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}\,.$$

The projection matrix $\boldsymbol{H} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top$ (which computes the orthogonal projection) is called the *hat matrix*, since it "puts the hat on $\boldsymbol{y}$".

The geometrical interpretation of the least-squares regression is illustrated in Figure 3.1 for $d = 1$ features, i.e., $\boldsymbol{X}$ is a matrix with two columns $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$. The least-squares regression function, i.e., the estimate $\hat{\boldsymbol{y}}$, is obtained by finding the orthogonal projection of the outcome vector $\boldsymbol{y}$ onto the subspace (the hyperplane) spanned by the vectors $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$. The projection $\hat{\boldsymbol{y}}$ represents the least squares estimate.

What if $\boldsymbol{X}$ is not full rank? In this case $\boldsymbol{X}^\top \boldsymbol{X}$ is a singular matrix and $\boldsymbol{w}^*$ is not uniquely defined, i.e., there exist multiple solutions that will give the same projection,

$$\exists \boldsymbol{\alpha} \neq \boldsymbol{w}^* \text{ such that } \hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^* = \boldsymbol{X}\boldsymbol{\alpha}\,.$$

The non-full-rank case occurs when one or more of the inputs are redundant, or when the number of inputs d is larger than the number of training examples N, which occurs, for example, in image analysis. This can be solved, for example, by the addition of a regularization term, which ensures that the matrix is non-singular.

3.1.2 Properties of parameter w^*

Let us analyze the properties of the optimal parameter vector $\boldsymbol{w}^*$. It is important to realize that the optimized parameter vector $\boldsymbol{w}^*$ depends on the training data. Since the training data is (normally) random, $\boldsymbol{w}^*$ is a random variable (or to be more precise, a vector of random variables). We clarify this in the following example.

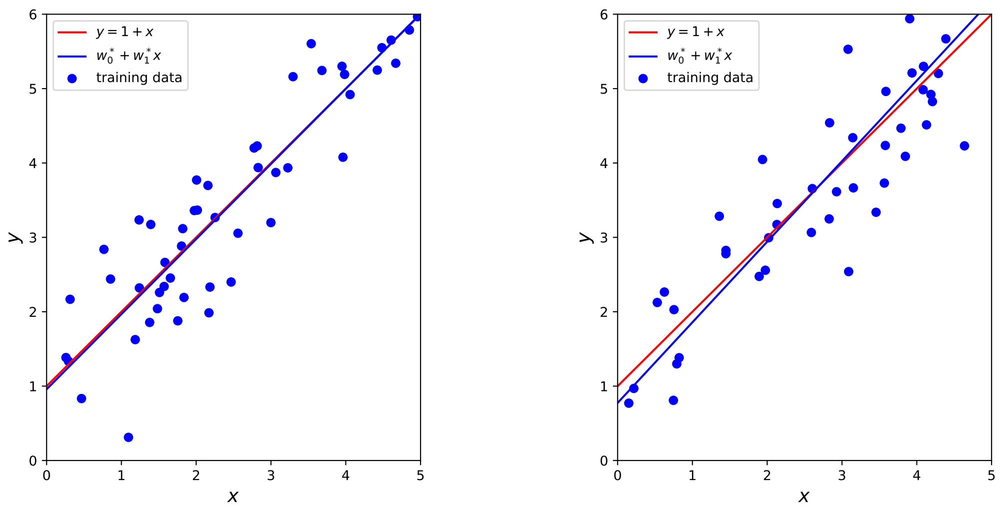

Figure 3.2: Two different training data sets (blue circles) generated from the function $y = 1 + x + \varepsilon$ with $\varepsilon = \mathcal{N}(0, 0.49)$. The red line corresponds to the true function $\tilde{f}(x) = 1 + x$, while the blue line corresponds to the learned function using the least-squares criterion.

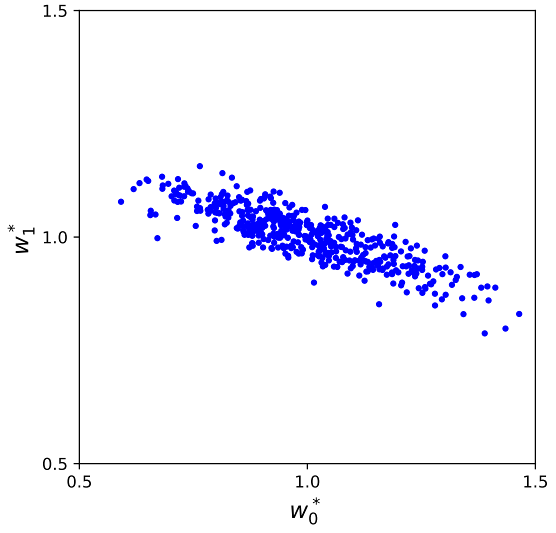

Figure 3.3: Values of $\boldsymbol{w}^*$ for 500 different training sets

Example 3.1 Consider a simple scenario where data is generated from the model

$$y = w_0 + w_1 x + \varepsilon\,,$$

with $\boldsymbol{w} = (1, 1)^\mathsf{T}$, i.e., $y = 1 + x + \varepsilon$, and $\varepsilon = \mathcal{N}(0, 0.49)$.

In Figure 3.2(left), we plot the the data set corresponding to $N = 50$ training examples (blue circles). The true function $y = 1 + x$ is depicted in red, while the blue curve corresponds to the learned function $y = w_0^* + w_1^* x$, where $\boldsymbol{w}^*$ is optimized using the least-squares criterion, i.e., by minimizing (3.2). Figure 3.2(right) depicts the same curves but for a different training set.

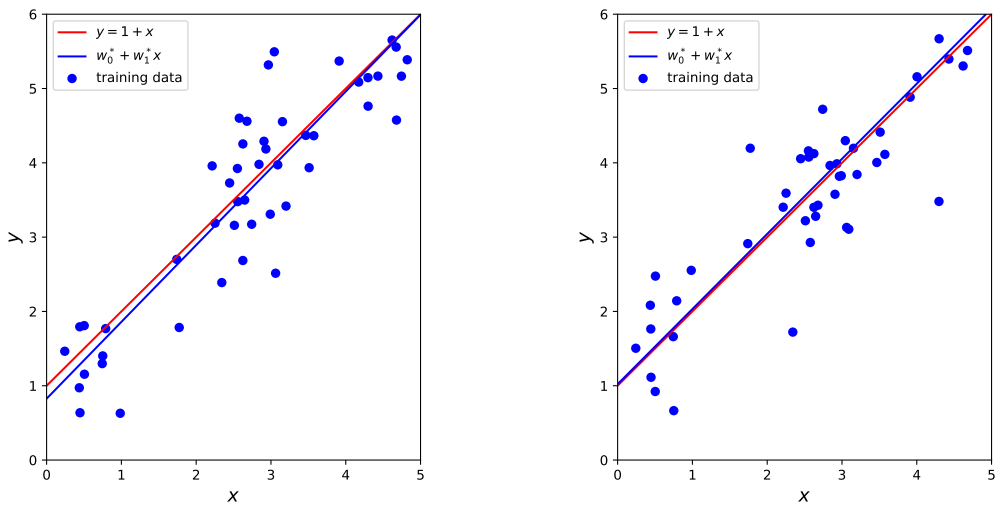

Figure 3.4: Training data set generated from two bivariate Gaussians of different mean (100 samples for each)

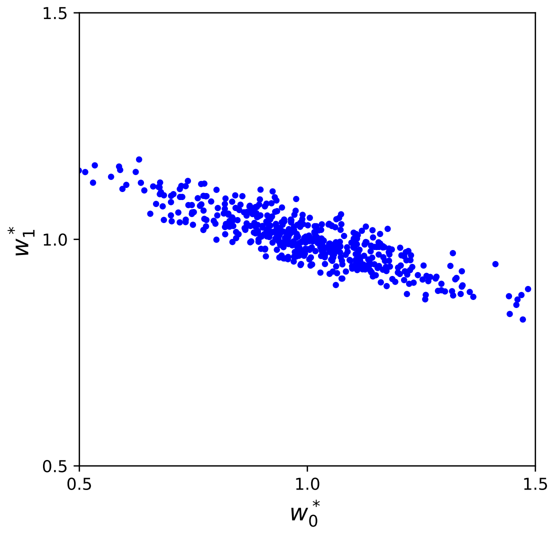

Figure 3.5: Values of $\boldsymbol{w}^*$ for 500 different training sets

As expected, different training data sets will yield a different vector $\boldsymbol{w}^*$. In Figure 3.3, we plot the estimated $\boldsymbol{w}^*$ for 500 different trials. Each training set $\mathcal{D}$ results in a different estimate $\boldsymbol{w}^*$.

In Figures 3.4 and 3.5, we show the same plots as in the previous examples with fixed training input data. As for the case where the input training data changes, the optimal $\boldsymbol{w}^*$ is different, due to the addition of the noise—the training data changes (as the y's are different)!

From the previous example, it is clear that we can regard $\boldsymbol{w}^*$ as a vector of random variables. What is its distribution? To answer this question we need to make some assumptions. In particular, we assume that the observations y_i are uncorrelated and have constant variance σ^2. We will also assume that the input training data $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$ is fixed.

We derive in the following the covariance matrix of $\boldsymbol{w}^*$. We first compute the expectation of $\boldsymbol{w}^*$,

$$\begin{aligned}\mathbb{E}[\boldsymbol{w}^*] &= \mathbb{E}\big[(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathbf{y}\big]\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathbb{E}[\mathbf{y}]\,.\end{aligned}$$

Now, recalling that $\mathbb{E}[y|\mathbf{x}=\boldsymbol{x}] = \tilde{f}(\boldsymbol{x}) = \boldsymbol{x}^\top\boldsymbol{w}$, it follows that, for fixed $\boldsymbol{X}$, $\mathbb{E}[\mathbf{y}] = \boldsymbol{X}\boldsymbol{w}$, and hence

$$\begin{aligned}\mathbb{E}[\boldsymbol{w}^*] &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w}\\ &= \boldsymbol{w}\,.\end{aligned}$$

Using this result, we get

$$\begin{aligned}\boldsymbol{w}^* - \mathbb{E}[\boldsymbol{w}^*] &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} - \boldsymbol{w}\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} - (\boldsymbol{X}^\top\boldsymbol{X})^{-1}(\boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{w}\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{\varepsilon}\,,\end{aligned}$$

where $\boldsymbol{\varepsilon}$ is a random column vector of dimension N.

The covariance matrix of $\boldsymbol{w}^*$ is computed as

$$\begin{aligned}\mathrm{Var}[\boldsymbol{w}^*] &= \mathbb{E}\big[(\boldsymbol{w}^* - \mathbb{E}[\boldsymbol{w}^*])(\boldsymbol{w}^* - \mathbb{E}[\boldsymbol{w}^*])^\top\big]\\ &= \mathbb{E}\big[(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\big]\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathrm{Var}[\boldsymbol{\varepsilon}]\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\,.\end{aligned}$$

If we assume that the entries of $\mathbf{y}$ are uncorrelated and all have the same variance of σ^2 (again given $\boldsymbol{X}$), then $\mathrm{Var}[\boldsymbol{\varepsilon}] = \sigma^2\boldsymbol{I}_N$, where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. With this, we obtain

$$\begin{aligned}\mathrm{Var}[\boldsymbol{w}^*] &= \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{I}_N\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\\ &= \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\\ &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\sigma^2\,.\end{aligned}$$

It remains to estimate the variance σ^2 (which is not known). It can be estimated using the sample variance,

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\,.$$

However, we typically estimate it as

$$\hat{\sigma}^2 = \frac{1}{N-d-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\,.$$

The reason is that using $N-d-1$ instead of N, makes $\hat{\sigma}^2$ an unbiased estimate of the true σ^2, i.e., $\mathbb{E}[\hat{\sigma}^2] = \sigma^2$.

We have now computed the mean and variance of $\boldsymbol{w}^*$. But what is its distribution? The vector $\boldsymbol{w}^*$ is distributed as

$$\boldsymbol{w}^* \sim \mathcal{N}(\boldsymbol{w}, (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\sigma^2)\,, \tag{3.6}$$

i.e., it is a multivariate normal distribution with mean vector $\boldsymbol{w}$ and covariance matrix $(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\sigma^2)$.

To show this, we use the fact that we are considering a linear regression model $y = \boldsymbol{x}^\mathsf{T}\boldsymbol{w} + \varepsilon$. The vector of outcomes $\boldsymbol{y}$ can thus be written as $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{\varepsilon}$. Hence, using (3.3),

$$\begin{aligned} \boldsymbol{w}^* &= (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y} \\ &= (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}(\boldsymbol{X}\boldsymbol{w} + \boldsymbol{\varepsilon}) \\ &= \boldsymbol{w} + (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{\varepsilon}\,, \end{aligned}$$

where $\boldsymbol{X}$ is fixed. The only source of randomness is hence $\boldsymbol{\varepsilon}$, which is a multivariate Gaussian random vector. Then, we can use the property that a linear transformation of a multivariate normal random vector also has a multivariate normal distribution, hence $\boldsymbol{w}^*$ is a multivariate normal distribution, with mean and variance as computed before.

3.1.3 Interpretation of the estimated weights

Machine learning can perform extremely well in many scenarios. However, a machine learning algorithm, particularly deep neural networks, is often a black box—one is unable to explain why the machine learning algorithm has produced a certain output. This makes machine learning difficult to adopt in highly regulated and critical domains such as healthcare, criminal justice, or autonomous driving.

Interpretability of machine learning algorithms is therefore crucial and there is significant research on *explainable* AI.

Linear regression allows such interpretability. Particularly, we are interested in giving some interpretation to the weights estimated by least squares. For example, it would be informative to determine which weights w_j of the true function (we recall that we assume that the true function is linear) are probably zero. This would help us understanding which features are irrelevant for the prediction—they are of little importance to explain the output—and therefore can be removed from the model.

If the true j-th parameter is zero, $w_j = 0$, then using (3.6) and the fact that the main diagonal of the covariance matrix contains variances (i.e., the covariance of each element with itself), the estimated parameter w_j^* is distributed as

$$w_j^* \sim \mathcal{N}(0, \sigma^2 v_j)\,,$$

where v_j is the j-th diagonal element of $(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}$.

Then if the actual value computed for w_j^* is larger than $\sigma^2 v_j$, then it is highly improbable that $w_j = 0$.

Statisticians have developed exact tests based on suitable distributions. To test the hypothesis that a particular coefficient $w_j = 0$, we form the standardized coefficient or so-called z-score,

$$z_j = \frac{w_j^*}{\hat{\sigma}\sqrt{v_j}}\,.$$

If $w_j = 0$, then z_j has a student's t-distribution (or simply t-distribution) with $N - d - 1$ degrees of freedom.

Example 3.2 Let's consider again the simple regression problem in Example 3.1 with a single feature x,

$$y = w_0 + w_1 x + \varepsilon\,,$$

where $\tilde{f}(\boldsymbol{x}) = w_0 + w_1 x$ is the true, unknown function. Assuming again $\boldsymbol{w} = (1, 1)^\mathsf{T}$, $N = 50$ training

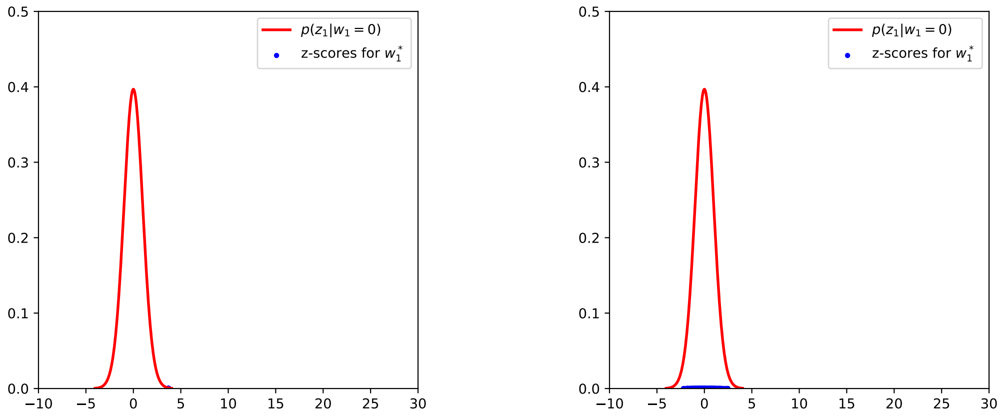

Figure 3.6: Distribution $p(z_1|w_1 = 0)$ (red curve) and the z-score of coefficient w_1^* estimated for 500 different training sets (blue circles). Left: $\boldsymbol{w} = (1, 1)^\mathsf{T}$, right: $\boldsymbol{w} = (3, 0)^\mathsf{T}$.

examples and $\sigma^2 = 0.49$, we plot in Figure 3.6(left) the distribution $p(z_1|w_1 = 0)$ (red curve), which is a t-distribution with $N - d - 1 = 48$ degrees of freedom. The figure also shows the z-score of coefficient w_1^* estimated for 500 different training sets (blue circles). Note that the computed z_1 scores lie in the tail of the t-distribution, which tells us that it is very improbable that w_1 is zero. Note that even if we didn't know w_1^* and only saw one training set $\mathcal{D}$, we would not think (rightly) that $w_1 = 0$.

Consider now the same regression problem but with $\boldsymbol{w} = (3, 0)^\mathsf{T}$, i.e., where $w_1 = 0$. The training examples, and the estimated curve $f(\boldsymbol{x}) = y = w_0^* + w_1^* x$ are depicted in Figure 3.7 for two different training sets. The values of w^* obtained for 500 different training sets are depicted in Figure 3.8. Note that in both cases, the coefficient w_1^* is not estimated as zero. In Figure 3.6(right) we plot the distribution $p(z_1|w_1 = 0)$ (red curve), which is the same t-distribution with $N - d - 1 = 48$ degrees of freedom as before, together with the z-score of coefficient w_1^* estimated for 500 different training sets (blue circles). Note that the computed z_1 scores lie well within the support of the t-distribution, which tells us that it is very likely that w_1 is zero (and indeed it is). Note that even if we didn't know $\boldsymbol{w}^*$ and only saw one training set, we would conclude in most trials that $w_1 \neq 0$. However, this statistical analysis allows us to conclude that with high probability $w_1 = 0$, and we could set it to zero in our model. Then we could re-learn the function $f(\boldsymbol{x})$.

Other tests can be performed. For example, we could test for the significance of groups of coefficients simultaneously, or get confidence bounds for w_j centered at w_j^*. In this course, however, we will not look into these.

3.2 The Gauss–Markov theorem

We will now discuss the Gauss–Markov theorem, which is a famous result in statistics. This theorem asserts that the least squares estimate $\boldsymbol{w}^*$ of the parameters $\boldsymbol{w}$ has the smallest variance among all linear unbiased estimates.

For simplicity, we consider the estimation of any linear combination of the parameters, i.e., $\theta = \boldsymbol{a}^\mathsf{T}\boldsymbol{w}$, a scalar. However, the result can be extended to the estimation of the entire parameter vector $\boldsymbol{w}$. Note that linear regression, $f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}\boldsymbol{w}$, is of this form.

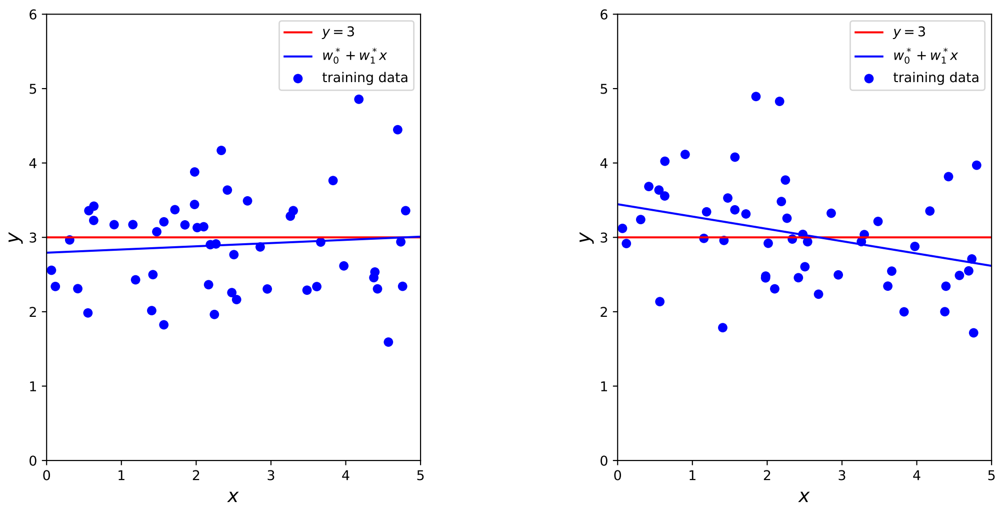

Figure 3.7: Two training data sets (blue circles) obtained from $y = 3 + \varepsilon$, and the estimated curve $f(\boldsymbol{x}) = y = w_0^* + w_1^* x$ (blue curve).

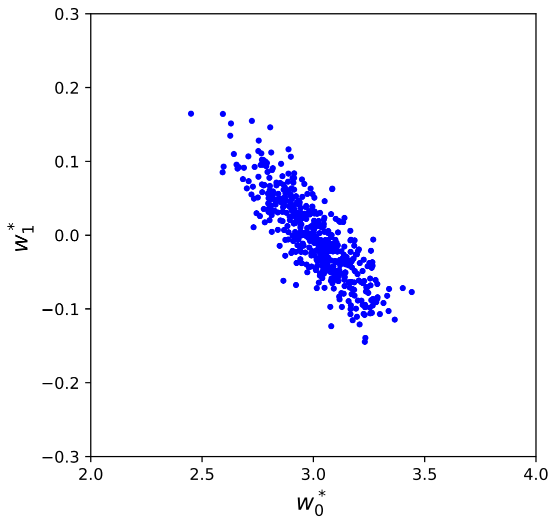

Figure 3.8: Values of $\boldsymbol{w}^*$ for 500 different training sets.

Assume the model $y = \boldsymbol{a}^\mathsf{T}\boldsymbol{w} + \varepsilon$. As we have seen, the least squares estimate of θ is (see (3.3))

$$\theta^* = \boldsymbol{a}^\mathsf{T}\boldsymbol{w}^* = \boldsymbol{a}^\mathsf{T}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}$$

If $\boldsymbol{X}$ is fixed, this is a linear function of the response vector $\boldsymbol{y}$.

Now, if the linear model $y = \boldsymbol{x}^\mathsf{T}\boldsymbol{w} + \varepsilon$ is correct, i.e., which implies $\mathbb{E}[\mathbf{y}|\boldsymbol{X}] = \boldsymbol{X}\boldsymbol{w}$, we obtain

$$\begin{aligned}\mathbb{E}[\boldsymbol{a}^\mathsf{T}\boldsymbol{w}^*] &= \mathbb{E}\big[\boldsymbol{a}^\mathsf{T}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\mathbf{y}\big]\\ &= \boldsymbol{a}^\mathsf{T}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\mathbb{E}[\mathbf{y}]\\ &= \boldsymbol{a}^\mathsf{T}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{w}\\ &= \boldsymbol{a}^\mathsf{T}\boldsymbol{w}\\ &= \theta\,.\end{aligned}$$

Hence, the estimation $\boldsymbol{a}^\mathsf{T}\boldsymbol{w}^*$ is unbiased.

We can now state the Gauss-Markov theorem, which we will give with no proof:

Theorem 3.1 (Gauss–Markov theorem) *Any other linear estimator $\tilde{\theta} = \boldsymbol{c}^\mathsf{T}\boldsymbol{y}$ that is unbiased for $\boldsymbol{a}^\mathsf{T}\boldsymbol{w}$, i.e., $\mathbb{E}[\boldsymbol{c}^\mathsf{T}\boldsymbol{y}] = \boldsymbol{a}^\mathsf{T}\boldsymbol{w} = \theta$, has variance*

$$Var[\boldsymbol{a}^\mathsf{T}\boldsymbol{w}^*] \leq Var[\boldsymbol{c}^\mathsf{T}\boldsymbol{y}]\,.$$

What's the implication of the Gauss–Markov theorem? We discuss this in the following.

Consider the mean-squared error of an estimator $\tilde{\theta}$ of θ,

$$\mathsf{MSE}(\tilde{\theta}) = \mathbb{E}\big[(\tilde{\theta} - \theta)^2\big]\,.$$

Expanding the quadratic term $(\tilde{\theta} - \theta)^2$, the MSE can be written as

$$\begin{aligned}\mathsf{MSE}(\tilde{\theta}) &= \mathbb{E}\big[(\tilde{\theta} - \theta)^2\big]\\ &= \mathbb{E}[(\tilde{\theta} - \mathbb{E}[\tilde{\theta}] + \mathbb{E}[\tilde{\theta}] - \theta)^2]\\ &= (\tilde{\theta} - \mathbb{E}[\tilde{\theta}])^2 + 2(\tilde{\theta} - \mathbb{E}[\tilde{\theta}])(\mathbb{E}[\tilde{\theta}] - \theta) + (\mathbb{E}[\tilde{\theta}] - \theta)^2\\ &\overset{(a)}{=} (\tilde{\theta} - \mathbb{E}[\tilde{\theta}])^2 + (\mathbb{E}[\tilde{\theta}] - \theta)^2\\ &= \underbrace{\mathrm{Var}[\tilde{\theta}]}_{\text{variance}} + (\underbrace{\mathbb{E}[\tilde{\theta}] - \theta}_{\text{bias}})^2\,,\end{aligned} \tag{3.7}$$

where (a) follows from the fact that $\mathbb{E}[\tilde{\theta}] - \theta = 0$ because the estimator is unbiased.

From (3.7), the Gauss–Markov theorem implies that, for all linear estimators with zero bias, the least-squares estimator has the smallest MSE.

Obviously there may be biased estimates which give a smaller MSE, i.e., they give a lower variance than the one of the least-squares estimator. In this cases we trade an increase in squared bias for a reduction in variance: the bias-variance trade-off again!

In practice, any model will introduce some deviations with respect the the true, unknown model. For example, we may assume that the true model is purely linear, but this will not hold in practice (it can be close to linear though). As a result, any model will be biased; picking the right model amounts to creating the right balance between bias and variance.

Note that we can relate the MSE with the prediction error. We already decomposed the prediction error as (see (2.27))

$$\begin{aligned}L(\hat{y}) &= \sigma^2 + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_\mathcal{D}[f(\boldsymbol{x})])^2 + \mathbb{E}_\mathcal{D}\big[(\mathbb{E}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\big]\\ &= \sigma^2 + (\underbrace{\tilde{f}(\boldsymbol{x}) - \mathbb{E}_\mathcal{D}[f(\boldsymbol{x})]}_{\text{bias}})^2 + \mathrm{Var}_\mathcal{D}[f(\boldsymbol{x})]\end{aligned}$$

with $f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}\boldsymbol{w}^*$.

In our setup here, $f(\boldsymbol{x})$ corresponds to $\tilde{\theta}$ and $\tilde{f}(\boldsymbol{x})$ to θ. Hence, the prediction error can be written as

$$\begin{aligned} L(\hat{y}) &= \sigma^2 + (\underbrace{\theta - \mathbb{E}[\tilde{\theta}]}_{\text{bias}})^2 + \mathrm{Var}_{\mathcal{D}}\left[\tilde{\theta}\right] \\ &= \sigma^2 + \mathsf{MSE}(\tilde{\theta}) \\ &= \sigma^2 + \mathsf{MSE}(f(\boldsymbol{x}))\,. \end{aligned}$$

Thus, the MSE is intimately related to prediction accuracy. They differ only by the constant σ^2, representing the variance of the new observation y. Note that the error component σ^2 is unrelated to what model is used to describe our data. It cannot be reduced for it exists in the true data generation process. The second source of error corresponding to the term $\mathsf{MSE}(f(\boldsymbol{x}))$ represents the error in the model and is under our control. Thus, based on the above expression, if we minimize the MSE of our estimator $f(\boldsymbol{x})$, we are effectively minimizing the expected (quadratic) prediction error which is our ultimate goal anyway

In short, we need to explore methods that seek to keep the total contribution of the two terms of the MSE (bias and variance) as small as possible by explicitly considering the trade-offs that come from methods that might increase one of the terms while decreasing the other.

3.3 Multiple regression

The linear model

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{d} x_j w_j, \tag{3.8}$$

that we have considered so far with $d > 1$ is called the multiple linear regression model. Let's take a look at the least squares estimates (3.3), through the lens of the estimates of a univariate (i.e., $d = 1$) linear model.

Consider first a univariate model with no intercept, i.e.,

$$y = wx + \varepsilon\,. \tag{3.9}$$

We consider a training data set with N training examples $\{\tilde{x}_i, y_i\}$, and define $\tilde{\boldsymbol{x}} = (\tilde{x}_1, \ldots, \tilde{x}_N)^\mathsf{T}$ the vector of training inputs (here we use $\tilde{\boldsymbol{x}}$ for the training set inputs so that we don't confuse this vector with the vector $\boldsymbol{x}$ of d features in (3.8)) and $\boldsymbol{y} = (y_1, \ldots, y_N)^\mathsf{T}$. We can write

$$\boldsymbol{y} = \boldsymbol{X}w + \boldsymbol{\varepsilon}$$

where $\boldsymbol{X}$ is now a single-column matrix, $\boldsymbol{X} = \tilde{\boldsymbol{x}}$.

The ordinary least square estimate of $\boldsymbol{w}$ are given by (3.3),

$$w^* = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}\,.$$

Since $\boldsymbol{X}$ is a single-column matrix, the products $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ and $\boldsymbol{X}^\mathsf{T}\boldsymbol{y}$ are scalars,

$$\boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \sum_{i=1}^{N} \tilde{x}_i^2 \quad \text{and} \quad \boldsymbol{X}^\mathsf{T}\boldsymbol{y} = \sum_{i=1}^{N} \tilde{x}_i y_i$$

so the least squares estimate of $\boldsymbol{w}$ is therefore given by

$$\begin{aligned} w^* &= \frac{\sum_{i=1}^N \tilde{x}_i y_i}{\sum_{i=1}^N \tilde{x}_i^2} \\ &= \frac{\tilde{\boldsymbol{x}}^\mathsf{T} \boldsymbol{y}}{\tilde{\boldsymbol{x}}^\mathsf{T} \tilde{\boldsymbol{x}}} \end{aligned}$$

$\tilde{\boldsymbol{x}}^\mathsf{T} \boldsymbol{y}$ and $\tilde{\boldsymbol{x}}^\mathsf{T} \tilde{\boldsymbol{x}}$ are inner products, so we may also write

$$w^* = \frac{\langle \tilde{\boldsymbol{x}}, \boldsymbol{y} \rangle}{\langle \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}} \rangle} . \tag{3.10}$$

The least squares residual for y_i is

$$r_i = y_i - \hat{y}_i = y_i - \tilde{x}_i w^* \tag{3.11}$$

and for the whole vector

$$\boldsymbol{r} = \boldsymbol{y} - \tilde{\boldsymbol{x}}^\mathsf{T} w^* \tag{3.12}$$

For the univariate regression to determine w^*, i.e., the optimal $\hat{y}$ under least-squares regression, we will say that we regress $\boldsymbol{y}$ onto $\tilde{\boldsymbol{x}}$, producing the coefficient (3.10).

We will see in the following that the univariate regression provides the building block for multiple linear regression. Consider the case of higher dimensions, i.e., more features, $p > 1$. In this case $\mathcal{D} = \{\boldsymbol{x}_i, y\}_i^N$ and the data matrix $\boldsymbol{X}$ is an $N \times p$ matrix (see (2.12) and recall that we assume no intercept). We denote the j-th column of $\boldsymbol{X}$ by $\tilde{\boldsymbol{x}}_j$, i.e., $\tilde{\boldsymbol{x}}_j$ is the vector of the N training inputs corresponding to feature j.

Assume that the columns of $\boldsymbol{X}$ are orthogonal, i.e., $\langle \tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_k \rangle = \tilde{\boldsymbol{x}}_j^\mathsf{T} \tilde{\boldsymbol{x}}_k = 0$ for all $j \neq k$. Then, it is easy to check that the multiple least squares estimates w_j^* are equal to

$$w_j^* = \frac{\langle \tilde{\boldsymbol{x}}_j, \boldsymbol{y} \rangle}{\langle \tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_j \rangle} ,$$

the univariate estimates. In other words, when the inputs are orthogonal, they have no effect on each other's parameter estimates in the model.

We see this in the following. We write down explicitly the outer product in the normal equations,

$$\begin{aligned} \boldsymbol{X}^\mathsf{T} \boldsymbol{X} &= \begin{pmatrix} \tilde{\boldsymbol{x}}_1^\mathsf{T} \\ \tilde{\boldsymbol{x}}_2^\mathsf{T} \\ \vdots \\ \tilde{\boldsymbol{x}}_d^\mathsf{T} \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{x}}_1 & \tilde{\boldsymbol{x}}_2 & \cdots & \tilde{\boldsymbol{x}}_d \end{pmatrix} \\ &= \begin{pmatrix} \tilde{\boldsymbol{x}}_1^\mathsf{T}\tilde{\boldsymbol{x}}_1 & \tilde{\boldsymbol{x}}_1^\mathsf{T}\tilde{\boldsymbol{x}}_2 & \cdots & \tilde{\boldsymbol{x}}_1^\mathsf{T}\tilde{\boldsymbol{x}}_d \\ \tilde{\boldsymbol{x}}_2^\mathsf{T}\tilde{\boldsymbol{x}}_1 & \tilde{\boldsymbol{x}}_2^\mathsf{T}\tilde{\boldsymbol{x}}_2 & \cdots & \tilde{\boldsymbol{x}}_2^\mathsf{T}\tilde{\boldsymbol{x}}_d \\ \vdots & \vdots & \cdots & \vdots \\ \tilde{\boldsymbol{x}}_d^\mathsf{T}\tilde{\boldsymbol{x}}_1 & \tilde{\boldsymbol{x}}_d^\mathsf{T}\tilde{\boldsymbol{x}}_2 & \cdots & \tilde{\boldsymbol{x}}_d^\mathsf{T}\tilde{\boldsymbol{x}}_d \end{pmatrix} \\ &= \begin{pmatrix} \tilde{\boldsymbol{x}}_1^\mathsf{T}\tilde{\boldsymbol{x}}_1 & 0 & \cdots & 0 \\ 0 & \tilde{\boldsymbol{x}}_2^\mathsf{T}\tilde{\boldsymbol{x}}_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \tilde{\boldsymbol{x}}_d^\mathsf{T}\tilde{\boldsymbol{x}}_d \end{pmatrix} . \end{aligned}$$

Using this, the estimate for $\boldsymbol{w}^*$ becomes

$$\begin{aligned}\boldsymbol{w}^* &= (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}\\ &= \begin{pmatrix} \tilde{\boldsymbol{x}}_1^\top \tilde{\boldsymbol{x}}_1 & 0 & \cdots & 0\\ 0 & \tilde{\boldsymbol{x}}_2^\top \tilde{\boldsymbol{x}}_2 & \cdots & 0\\ \vdots & \vdots & \cdots & \vdots\\ 0 & 0 & \cdots & \tilde{\boldsymbol{x}}_d^\top \tilde{\boldsymbol{x}}_d \end{pmatrix}^{-1} (\boldsymbol{X}^\top \boldsymbol{y})\\ &= \begin{pmatrix} \tilde{\boldsymbol{x}}_1^\top \tilde{\boldsymbol{x}}_1 & 0 & \cdots & 0\\ 0 & \tilde{\boldsymbol{x}}_2^\top \tilde{\boldsymbol{x}}_2 & \cdots & 0\\ \vdots & \vdots & \cdots & \vdots\\ 0 & 0 & \cdots & \tilde{\boldsymbol{x}}_d^\top \tilde{\boldsymbol{x}}_d \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\boldsymbol{x}}_1^\top \boldsymbol{y}\\ \tilde{\boldsymbol{x}}_2^\top \boldsymbol{y}\\ \vdots\\ \tilde{\boldsymbol{x}}_d^\top \boldsymbol{y}\end{pmatrix}\\ &= \begin{pmatrix} \frac{\tilde{\boldsymbol{x}}_1^\top \boldsymbol{y}}{\tilde{\boldsymbol{x}}_1^\top \tilde{\boldsymbol{x}}_1}\\ \frac{\tilde{\boldsymbol{x}}_2^\top \boldsymbol{y}}{\tilde{\boldsymbol{x}}_2^\top \tilde{\boldsymbol{x}}_2}\\ \vdots\\ \frac{\tilde{\boldsymbol{x}}_d^\top \boldsymbol{y}}{\tilde{\boldsymbol{x}}_d^\top \tilde{\boldsymbol{x}}_d}\end{pmatrix},\end{aligned}$$

i.e., each beta is obtained as in the univariate case,

$$w_j^* = \frac{\langle \tilde{\boldsymbol{x}}_j, \boldsymbol{y}\rangle}{\langle \tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_j\rangle}\,,$$

Thus, when the feature vectors are orthogonal they have no effect on each other.

Unfortunately, data acquired from observations is rarely orthogonal. Hence, to take advantage of the nice property when inputs are orthogonal, we will have to orthogonalize the columns $\tilde{\boldsymbol{x}}_j$.

To see how we can do this orthogonalization, consider again the univariate case but suppose that now have an intercept, i.e, we assume the model

$$y = w_0 + w_1 x + \varepsilon\,. \tag{3.13}$$

In this case the data matrix $\boldsymbol{X}$ is a two-column matrix with the first column with all ones. In this case, the least squares coefficient w_1^* has the form

$$w_1^* = \frac{\langle \tilde{\boldsymbol{x}} - \bar{x}\mathbf{1}, \boldsymbol{y}\rangle}{\langle \tilde{\boldsymbol{x}} - \bar{x}\mathbf{1}, \tilde{\boldsymbol{x}} - \bar{x}\mathbf{1}\rangle}\,, \tag{3.14}$$

where $\bar{x} = \sum_{i=1}^N \frac{\tilde{x}_i}{N}$ and $\mathbf{1}$ is the vector of all ones corresponding to the first column of $\boldsymbol{X}$

Now observe that the estimate (3.14) of the coefficient w_1^* can be obtained as the result of the application of two univariate regressions with no intercept (see (3.10)) in succession:

- Regress $\tilde{\boldsymbol{x}}$ onto $\mathbf{1}$, producing the coefficient

$$w_{(\mathbf{1})}^* = \frac{\langle \mathbf{1}, \tilde{\boldsymbol{x}}\rangle}{\langle \mathbf{1}, \mathbf{1}\rangle} = \frac{\sum_{i=1}^N \tilde{x}_i}{N} = \bar{x}\,.$$

  and obtaining the residual $\boldsymbol{z} = \tilde{\boldsymbol{x}} - w_{(\mathbf{1})}^*\mathbf{1} = \tilde{\boldsymbol{x}} - \bar{x}\mathbf{1}$;

- Regress $\boldsymbol{y}$ onto the residual $\boldsymbol{z}$ to give the coefficient w_1^*.

Note that step 1 orthogonalized $\tilde{\boldsymbol{x}}$ with respect to $\mathbf{1}$ (see Appendix for the known Gram-Schmidt orthogonalization of vectors). Step 2 is just univariate regression, using the orthogonal predictors $\mathbf{1}$ and $\boldsymbol{z}$.

This recipe can be generalized to the case of d features and is given in the algorithm below, which is called *regression by successive orthogonalization* or Gram-Schmidt for multiple regression.

Algorithm (Regression by successive orthogonalization)

1. Initialize $\boldsymbol{z}_0 = \tilde{\boldsymbol{x}}_0 = \mathbf{1}$
2. For $j = 1, \dots, p$

   Regress $\tilde{\boldsymbol{x}}_j$ onto $\boldsymbol{z}_0, \boldsymbol{z}_1, \dots, \boldsymbol{z}_{j-1}$ to obtain the coefficients $\gamma_{\ell,j} = \langle \boldsymbol{z}_\ell, \tilde{\boldsymbol{x}}_j \rangle / \langle \boldsymbol{z}_\ell, \boldsymbol{z}_\ell \rangle$, $\ell = 0, \dots, j-1$ and residual vector $\boldsymbol{z}_j = \tilde{\boldsymbol{x}}_j - \sum_{i=0}^{j-1} \gamma_{i,j} \boldsymbol{z}_i$
3. Regress $\boldsymbol{y}$ on the residual $\boldsymbol{z}_d$ to give the estimate w_d^* given by

$$w_d^* = \frac{\langle \boldsymbol{z}_d, \boldsymbol{y} \rangle}{\langle \boldsymbol{z}_d, \boldsymbol{z}_d \rangle} \,.$$

If we did this $d+1$ times, changing the ordering so that each of the $\tilde{\boldsymbol{x}}_j$ came last, we would obtain all least squares coefficients w_j.

3.4 Multiple outputs

Up to now, we have considered the case of a single output variable y. In some applications, however, we may wish to predict $K > 1$ output variables, which we will collect in the output vector $\boldsymbol{y} = (y_1, \dots, y_K)^\mathsf{T}$, from the input $\boldsymbol{x} = (x_1, \dots, x_d)^\mathsf{T}$. We consider the case of multiple outputs in the following.

We assume a linear model for each output,

$$\begin{aligned} y_j &= w_{j,0} + \sum_{\ell=1}^{d} x_\ell w_{j,\ell} + \varepsilon_i \\ &= w_{j,0} + \boldsymbol{w}_j^\mathsf{T} \boldsymbol{x} + \varepsilon_i \\ &= f_j(\boldsymbol{x}) + \varepsilon_i \,. \end{aligned}$$

where $\boldsymbol{w}_j = (w_{j,1}, \dots, w_{j,d})^\mathsf{T}$.

Considering a data set with N training examples, $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, where each output $\boldsymbol{y}_i$ is now a vector of length K, $\boldsymbol{y}_i = (y_{i,1}, \dots, y_{i,K})$, we can build the $N \times K$ matrix of outputs $\boldsymbol{Y}$, whose i-th row corresponds to the length-K output vector $\boldsymbol{y}_i$, and write the linear regression as

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{E} \,,$$

where $\boldsymbol{X}$ is the $N \times (d+1)$ data matrix with $d+1$ columns corresponding to the d features and a leading column of ones corresponding to the intercept, given in (2.12), $\boldsymbol{W}$ is the $(d+1) \times K$ matrix which collects all coefficients w,

$$\boldsymbol{W} = \begin{pmatrix} w_{1,0} & w_{2,0} & \dots & w_{K,0} \\ w_{1,1} & w_{2,1} & \dots & w_{K,1} \\ \vdots & \vdots & \vdots & \vdots \\ w_{1,d} & w_{2,d} & \dots & w_{K,d} \end{pmatrix} ,$$

and $\boldsymbol{E}$ is the $N \times K$ matrix of errors.

We can generalize the loss function (3.2) for the univariate case to multiple outputs as

$$\begin{aligned}
\mathsf{RSS}(\boldsymbol{W}) &= \sum_{i=1}^{N}\sum_{j=1}^{K}(y_{i,j} - f_j(\boldsymbol{x}_i))^2 \\
&= \sum_{i=1}^{N}\sum_{j=1}^{K}\left(y_{i,j} - \left(w_{j,0} + \sum_{\ell=1}^{d} x_{i,\ell} w_{j,\ell}\right)\right)^2 \\
&= \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{W}\|_{\mathsf{F}}^2 \,,
\end{aligned}$$

where $\|\boldsymbol{A}\|_{\mathsf{F}}^2$ is the Frobenius norm,

$$\|\boldsymbol{A}\|_{\mathsf{F}}^2 = \sum_i \sum_j a_{i,j}^2 \,.$$

The least squares solution has exactly the same form as before,

$$\begin{aligned}
\boldsymbol{W}^* &= \arg\min_{\boldsymbol{W}} \ \mathsf{RSS}(\boldsymbol{W}) \\
&= (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{Y} \,. \qquad (3.15)
\end{aligned}$$

and the fit is given by

$$\hat{\boldsymbol{Y}}^* = \boldsymbol{X}\boldsymbol{W}^* = \boldsymbol{X}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{Y} \,.$$

Thus, the coefficients for the i-th output $y_{1,i}, \ldots, y_{N,i}$ correpsond to the solution to the least squares regression of $y_{1,i}, \ldots, y_{N,i}$ on the columns of $\boldsymbol{X}$, i.e., multiple outputs do not affect one another's least squares estimates.

3.5 Subset selection

The least squares estimates for linear regression often have low bias but high variance, which can affect the prediction accuracy. Accuracy can sometimes be improved by setting some coefficients w_j to zero. This increases the bias, but reduces the variance and in turn may improve the accuracy of the prediction.

Hence, it would be interesting to determine a smaller subset of features which is the most informative (i.e., captures the main properties of the data). This will increase the interpretability of the data and in turn, improve accuracy. In particular, we will be interested in finding the set of k features that are most informative, which is typically referred to as variable subset selection. Least squares regression can then be applied on the k features retained to estimate the coefficients of the model.

In the following, we describe a number of approaches to variable subset selection for linear regression. We can think of this approach as constraining the number of non-zero coefficients in the model.

3.5.1 Best-subset selection

Best subset regression finds for each $k \in \{1, \ldots, p\}$ the subset of size k that gives the smallest RSS (3.2)

$$\mathsf{RSS}_{\mathsf{BSS}}(j_1, \ldots, j_k) = \min_{w_0, w_{j_1} \ldots, w_{j_k}} \sum_{i=1}^{N}\left(y_i - w_0 - \sum_{\ell=1}^{k} w_{j_\ell} x_{i,j_\ell}\right)^2 \,,$$

with $j_i \in \{1, \dots, d\}$.

There are $\binom{d}{k}$ different subsets to try for a given k. If $d \leq 40$, there exist computationally feasible algorithms for finding the best subsets. For larger d (and if k is not small), however, the complexity to solve this problem becomes infeasible. In this case, one can resort to various heuristics for solving this approximately: forward-stepwise selection, backward-stepwise selection, and forward-stagewise regression.

A question still remains of how to choose the best value of k. Once again, the best value for k will be a trade-off between bias and variance.

Remark 3.1 Subset selection is in general not considered a favorable modern approach compared to other approaches such as ridge regression and (especially) lasso, which we discuss in Sections 3.6 and 3.7, respectively. Specifically, ridge and lasso are convex formulations, and hence can be solved efficiently even in high dimensions.

3.5.2 Forward- and backward-stepwise selection

Instead of searching all possible subsets (infeasible for large d), we can consider a greedy approach.

Forward-stepwise selection starts with the intercept, and then sequentially augments the model with the predictor that most improves the fit. Formally,

$$j_\ell = \arg\min_{j \in \mathcal{I}} \min_{w_0, w_{j_1}, \dots, w_{j_{\ell-1}}, w_j} \sum_{i=1}^{N} \left(y_i - w_0 - \sum_{m=1}^{\ell-1} w_{j_m} x_{i,j_m} - w_j x_{i,j} \right)^2,$$

where $\mathcal{I} = \{1, \dots, p\} \backslash \{j_1, \dots, j_{\ell-1}\}$.

Forward-stepwise may be suboptimal compared to the best subset selection but may be preferred because

- It is computational feasible for large d
- Best subset selection may overfit
- Forward stepwise will probably produce a function with lower variance but perhaps more bias

Backward-stepwise selection starts with the complete model, and sequentially deletes the predictor that contributes least to the fit. Particularly, at each step of the algorithm, we remove the variable with smallest z-score, i.e., the one whose coefficient w_j is most likely to be zero.

3.6 Shrinkage methods: Ridge regression

Subset selection essentially involves setting some of the w_j's to 0, and letting the others be estimated using least squares. Selecting a subset of predictors produces a model that is interpretable and probably has lower prediction error than the full model. The drawback of best-subset selection is that it is a discrete procedure (variables are either retained or discarded) and there are too many required subsets to search over. Furthermore, because it is a discrete process, it often exhibits high variance.

A less drastic approach is to allow all estimates to be positive, but to constrain them in some way so that they do not become too big. Methods that impose a penalty on the estimates to avoid them becoming too big are known as *regularization methods*. One of them is ridge regression, which we discuss next.

Ridge regression shrinks the regression coefficients w_j's by imposing a penalty on their size. The ridge coefficients are obtained by minimizing

$$\boldsymbol{w}^*_{\mathsf{ridge}} = \arg\min_{\boldsymbol{w}} \underbrace{\sum_{i=1}^{N}\left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j} w_j\right)^2}_{\mathsf{RSS}} + \lambda \underbrace{\sum_{j=1}^{d} w_j^2}_{\text{reg. function}} \tag{3.16}$$

$$= \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\tilde{\boldsymbol{w}}\|^2 \,. \tag{3.17}$$

Observe that ridge regression modifies the least squares criterion by adding the quadratic regularization function $\sum_{j=1}^{d} w_j^2$ multiplied by λ, where $\lambda \geq 0$ is a parameter that controls the amount of shrinkage. As λ increases, more weight is placed on the w_j's being small, i.e., for larger λ, the coefficients are more shrink toward zero. For the extreme case $\lambda = \infty$, we get $\tilde{\boldsymbol{w}}^*_{\mathsf{ridge}} = 0$. For $\lambda = 0$, we recover the conventional linear regression. For λ in between, we are balancing two ideas: fitting a linear model of y on $\boldsymbol{X}$, and shrinking the coefficients.

Remark 3.2 The intercept w_0 is typically not included in the penalty term, as seen in (3.16). The reason for this is that penalizing the intercept makes the procedure dependent on the origin chosen for $\boldsymbol{y}$—we could add a constant amount c to each of the targets y_i, and this would not result in a shift of the predictions by the same amount c.

Remark 3.3 Note that if we rescale the columns of $\boldsymbol{X}$, then the least squares model is not affected. However, having different columns with different scale is critical to penalty approaches, and so it is typical to standardize columns of $\boldsymbol{X}$ before applying ridge regression. A variable is standardized by subtracting from it its sample mean and by dividing it by its standard deviation. After being standardized, the variable has zero mean and unit standard deviation. The idea is to make sure that the features are on a similar scale.

We can rewrite (3.16) equivalently as

$$\boldsymbol{w}^*_{\mathsf{ridge}} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N}\left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j} w_j\right)^2 \tag{3.18}$$

$$\text{subject to } \|\tilde{\boldsymbol{w}}\|^2 \leq t \tag{3.19}$$

which puts an explicit constraint on the size of parameters $\{w_j\}$. There is a one-to-one correspondence between the parameters λ and t: The regularization function forces the norm of the solution $\tilde{\boldsymbol{w}}$ (see (3.17)) to be small, particularly with larger values of the hyperparameter λ, which corresponds to the constraint (3.19).

It can be shown ([1, Ex. 3.5]) that the ridge regression problem (3.16) is equivalent to the problem

$$\boldsymbol{w}^*_{\mathsf{c}} = \arg\min_{\boldsymbol{w}_{\mathsf{c}}} \sum_{i=1}^{N}\left(y_i - w_{\mathsf{c},0} - \sum_{i=1}^{d} \tilde{x}_{i,j} w_{\mathsf{c},j}\right)^2 + \lambda \sum_{j=1}^{d} w_{\mathsf{c},j}^2 \,, \tag{3.20}$$

where

$$\tilde{x}_{i,j} = x_{i,j} - \bar{x}_j = x_{i,j} - \frac{1}{N}\sum_{\ell=1}^{N} x_{\ell,j}$$

is the centered version of the input data, i.e., the data is shifted to have zero mean, and then the solution to (3.20)—and hence to (3.16)—can be obtained in two steps: First, we can fit the intercept $w_{\mathsf{c},0}$ separately, and then the remaining coefficients can be estimated by a ridge regression without intercept, using the centered $\tilde{x}_{i,j}$.

Fitting $w_{\mathsf{c},0}$ separately yields

$$w^*_{\mathsf{c},0} = \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \,.$$

This is obtained by setting the derivative with respect to $w_{\mathsf{c},0}$ to zero.

Assuming centered inputs $\tilde{x}_{i,j} = x_{i,j} - \bar{x}_j$ and $\tilde{y}_i = y_i - \bar{y}$, the remaining coefficients $w_{\mathsf{c},j}$, $j = 1, \ldots, p$, are obtained applying ridge regression without an intercept (we remove the tildes for notational simplicity)

$$\begin{aligned} \boldsymbol{w}^*_{\mathsf{c}} &= \arg\min_{\boldsymbol{w}_{\mathsf{c}}} \sum_{i=1}^{N}\left(y_i - \sum_{i=1}^{d} x_{i,j} w_{\mathsf{c},j}\right)^2 + \lambda \sum_{j=1}^{d} w^2_{\mathsf{c},j} \\ &= \arg\min_{\boldsymbol{w}_{\mathsf{c}}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}_{\mathsf{c}})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}_{\mathsf{c}}) + \lambda\|\boldsymbol{w}_{\mathsf{c}}\|^2 \,, \end{aligned} \tag{3.21}$$

where $\boldsymbol{y} = (y_1, \ldots, y_N)^{\mathsf{T}}$, $\boldsymbol{w}_{\mathsf{c}} = (w_{\mathsf{c},1}, \ldots, w_{\mathsf{c},d})^{\mathsf{T}}$, and $\boldsymbol{X}$ is now an $N \times p$ matrix

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ \vdots & \vdots & \ldots & \vdots \\ x_{N,1} & x_{N,2} & \ldots & x_{N,d} \end{pmatrix} .$$

The ridge regression solution to (3.21) (with centered inputs) is

$$\boldsymbol{w}^*_{\mathsf{ridge}} = (\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathsf{T}}\boldsymbol{y} \,,$$

where $\boldsymbol{I}$ is a $d \times d$ identity matrix.

Note that in ridge regression, the problem of the linear regression (3.3) of inverting the potentially singular matrix $\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X}$ is averted as $\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X} + \lambda\boldsymbol{I}$ is full rank even if $\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X}$ is not.

3.6.1 Insight into ridge regression

We will now try to get some more insight into ridge regression by considering the singular value decomposition (SVD) of the centered input matrix $\boldsymbol{X}$ (without the intercept). The SVD of the $N \times d$ matrix $\boldsymbol{X}$ has the form

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathsf{T}} \,, \tag{3.22}$$

where $\boldsymbol{U}$ is an $N \times d$ semi-orthogonal matrix whose columns span the column space of $\boldsymbol{X}$ and $\boldsymbol{V}$ is a $d \times d$ orthogonal matrix (see Appendix B.3). $\boldsymbol{D}$ is a $d \times d$ diagonal matrix with diagonal entries $d_1 \geq d_2 \geq \ldots \geq d_d$ being the singular values of $\boldsymbol{X}$. If one or more values $d_j = 0$, $\boldsymbol{X}$ is singular.

Using the singular value decomposition, we can write the least squares fitted vector as (see (3.5))

$$\boldsymbol{X}\boldsymbol{w}^{\mathsf{ls}} = \boldsymbol{X}(\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathsf{T}}\boldsymbol{y} \tag{3.23}$$

$$= \boldsymbol{U}\boldsymbol{U}^{\mathsf{T}}\boldsymbol{y} \,. \tag{3.24}$$

We obtain this result as follows. First, note that

$$\begin{aligned}\boldsymbol{X}^\mathsf{T}\boldsymbol{X} &= \boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\mathsf{T}\\ &= \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\mathsf{T}\,.\end{aligned}$$

Also, using (B.2),

$$(\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\mathsf{T})^{-1} = (\boldsymbol{V}^\mathsf{T})^{-1}\boldsymbol{D}^{-2}\boldsymbol{V}^{-1}\,.$$

Using this result in (3.23) we obtain

$$\begin{aligned}\boldsymbol{X}\boldsymbol{w}^{\mathsf{ls}} &= \boldsymbol{X}(\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{X}(\boldsymbol{V}^\mathsf{T})^{-1}\boldsymbol{D}^{-2}\boldsymbol{V}^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\mathsf{T}(\boldsymbol{V}^\mathsf{T})^{-1}\boldsymbol{D}^{-2}\boldsymbol{V}^{-1}(\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\mathsf{T})^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\mathsf{T}(\boldsymbol{V}^\mathsf{T})^{-1}\boldsymbol{D}^{-2}\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{D}^\mathsf{T}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{D}^{-2}\boldsymbol{D}^\mathsf{T}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{D}^{-2}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{U}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= \sum_{i=1}^{d}\boldsymbol{u}_i(\boldsymbol{u}_i^\mathsf{T}\boldsymbol{y})\,,\end{aligned}\tag{3.25}$$

where $\boldsymbol{u}_i$ are the columns of $\boldsymbol{U}$.

Note that $\boldsymbol{U}^\mathsf{T}\boldsymbol{y}$ are the coordinates of $\boldsymbol{y}$ with respect to the orthonormal basis $\boldsymbol{U}$, i.e., the least squares solution $\boldsymbol{U}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}$ is the closest approximation to $\boldsymbol{y}$ in the subspace spanned by the columns of $\boldsymbol{U}$ (column space of $\boldsymbol{X}$).

To compare how the fitted values $\hat{\boldsymbol{y}}$ obtained in ridge regression compare with ordinary least squares we next consider the SVD expression for $\boldsymbol{w}^*_{\mathsf{ridge}}$,

$$\begin{aligned}\boldsymbol{w}^*_{\mathsf{ridge}} &= (\boldsymbol{X}^\mathsf{T}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}\\ &= (\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\mathsf{T}+\lambda\boldsymbol{V}\boldsymbol{V}^\mathsf{T})^{-1}\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= (\boldsymbol{V}(\boldsymbol{D}^2+\lambda\boldsymbol{I})\boldsymbol{V}^\mathsf{T})^{-1}\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= (\boldsymbol{V}^\mathsf{T})^{-1}(\boldsymbol{D}^2+\lambda\boldsymbol{I})^{-1}\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{V}(\boldsymbol{D}^2+\lambda\boldsymbol{I})^{-1}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\,.\end{aligned}$$

Using this, the output generated by ridge regression is

$$\begin{aligned}\hat{\boldsymbol{y}}_{\mathsf{ridge}} &= \boldsymbol{X}\boldsymbol{w}^*_{\mathsf{ridge}}\\ &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\mathsf{T}\boldsymbol{V}(\boldsymbol{D}^2+\lambda\boldsymbol{I})^{-1}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\\ &= \boldsymbol{U}\boldsymbol{D}(\boldsymbol{D}^2+\lambda\boldsymbol{I})^{-1}\boldsymbol{D}\boldsymbol{U}^\mathsf{T}\boldsymbol{y}\,.\end{aligned}\tag{3.26}$$

Now note that in this last expression $\boldsymbol{D}(\boldsymbol{D}^2+\lambda\boldsymbol{I})^{-1}\boldsymbol{D}$ is a diagonal matrix with elements given by $\frac{d_i^2}{d_i^2+\lambda}$ and the vector $\boldsymbol{U}^\mathsf{T}\boldsymbol{y}$ is the coordinates of the vector $\boldsymbol{y}$ in the basis spanned by the d-columns of $\boldsymbol{U}$. So, like linear regression, ridge regression computes the coordinates of $\boldsymbol{y}$ with respect to the orthonormal basis $\boldsymbol{U}$. Thus, writing the expression given by (3.26) by summing columns we obtain

$$\hat{\boldsymbol{y}}_{\mathsf{ridge}} = \sum_{i=1}^{d}\boldsymbol{u}_i\frac{d_i^2}{d_i^2+\lambda}\boldsymbol{u}_i^\mathsf{T}\boldsymbol{y}\,.\tag{3.27}$$

Note that this result is similar to that found in (3.25) derived for ordinary least squares regression but in ridge-regression the inner products $\boldsymbol{u}_i^\top \boldsymbol{y}$ are now scaled by the factors $\frac{d_i^2}{d_i^2+\lambda}$ (as $\lambda \geq 0$, $\frac{d_i^2}{d_i^2+\lambda} \leq 1$). Also, note that more shrinkage is applied to basis vectors with smaller d_i^2.

3.6.2 Principal components and meaning of the d_i^2's

Let's interpret what a small value of d_i^2 means. Consider a collection of points in $\mathbb{R}^d$, i.e., a collection of d-dimensional vectors and assume that we want to summarize them by projecting these points down onto a q-dimensional subspace, and we would like to find the most informative summary. In other words, we are looking for directions $\boldsymbol{b} \in \mathbb{R}^d$ with large spread of the data.

The $q \leq p$ principal components of these sets of vectors are the q orthogonal directions in space along which projections of the original vectors have largest variance. The principal components capture the direction in which the data varies most.

Rather than maximizing variance, it might sound more reasonable to look for the directions with the smallest average (squared) distance between the original vectors and their projections onto these directions; this turns out to be equivalent to maximizing the variance. Hence, we can see the principal components of a sequence of q (unit) vectors, where the i-th vector is the direction of a line that best fits the data while being orthogonal to the first $i-1$ vectors. And here, a best-fitting line is defined as one that minimizes the average squared distance from the points to the line.

Hence, the first principal component is the direction in space along which projections have the largest variance. The second principal component is the direction which maximizes variance among all directions orthogonal to the first. In general, the k-th principal component is the variance-maximizing direction orthogonal to the previous $k-1$ components. In total, there are d components, of which we consider the q principal ones.

Example 3.3 In Figure 3.9, the circles represent a two-dimensional data set in the x–y coordinate system. The principal direction in which the data varies is shown by the $\boldsymbol{u}$ axis and the second most important direction is the $\boldsymbol{v}$ axis, orthogonal to it. If we place the $\boldsymbol{u}$–$\boldsymbol{v}$ axis system at the mean of the data it gives us a compact representation. If we transform each (x, y) coordinate into its corresponding (u, v) value, the data is de-correlated, meaning that the covariance between the u and v variables is zero. For a given set of data, the principal component analysis finds the axis system defined by the principal directions of variance (i.e., the u–v axis system the figure). The directions $\boldsymbol{u}$ and $\boldsymbol{v}$ are called the principal components.

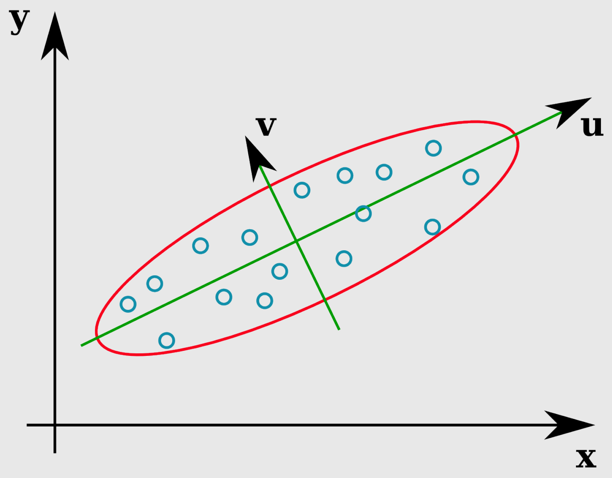

Figure 3.9: Principal components.

The principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on.

Now let's go back to our regression problem. Consider again the centered matrix $\boldsymbol{X}$. The sample covariance matrix of the data is given by (see (B.1))

$$\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\,.$$

From the SVD of $\boldsymbol{X}$ in (3.22), it follows that

$$\boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\mathsf{T}\,,$$

which is the eigendecomposition of $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$. The columns of $\boldsymbol{V}$, $\boldsymbol{v}_1,\dots,\boldsymbol{v}_d$, are the eigenvectors of $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ and therefore the principal components of $\boldsymbol{X}$, and the diagonal entries of $\boldsymbol{D}^2$, $d_1^2 \geq d_2^2 \geq \ldots \geq d_p^2$, are the corresponding eigenvalues.

Now, project the input of each training example onto the first principal component direction $\boldsymbol{v}_1$ to get $z_i^{(1)} = \boldsymbol{v}_1^\mathsf{T}\boldsymbol{x}_i$. The variance of the $z_i^{(1)}$'s is given by (remember that the $\boldsymbol{x}_i$'s are centred)

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}\left(z_i^{(1)}\right)^2 &= \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{v}_1^\mathsf{T}\boldsymbol{x}_i\boldsymbol{x}_i^\mathsf{T}\boldsymbol{v}_1\\ &= \frac{1}{N}\boldsymbol{v}_1^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{v}_1\\ &= \frac{1}{N}\boldsymbol{v}_1^\mathsf{T}\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\mathsf{T}\boldsymbol{v}_1\\ &= \frac{d_1^2}{N}\,, \end{aligned} \tag{3.28}$$

where the last equality follows from the fact that the eigenvectors are orthogonal and of norm one (since the columns of $\boldsymbol{V}$ are an orthonormal basis of $\mathbb{R}^d$, i.e., the columns are unit vectors (vectors of length 1—or norm 1—and orthogonal to each other).

Note that as d_1 is the largest eigenvalue, the variance of the projection $z_i^{(1)}$, given in (3.28), is the largest, hence $\boldsymbol{v}_1$ represents the direction (of unit length) onto which the projected points have largest variance. The projection $\boldsymbol{z}_1 = \boldsymbol{X}\boldsymbol{v}_1$ is called the first principal component of $\boldsymbol{X}$.

The subsequent projections $z_i^{(j)}$, $j = 2, \dots, p$, which are orthogonal to the preceding ones, have decreasing variance d_j^2/N. The last principal component, corresponding to $\boldsymbol{v}_p^\mathsf{T}$ and d_p, has minimum variance. Hence, the small d_j correspond to the directions of the column space of $\boldsymbol{X}$ having small variance. From (3.27), these are precisely the directions that ridge regression shrinks the most! Therefore, ridge regression implicitly assumes that the output will vary most in the directions of high variance of the inputs (a reasonable assumption, but not always true). These directions constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

3.7 Shrinkage methods: The lasso

The lasso (least absolute shrinkage and selection operator) estimate is defined by

$$\boldsymbol{w}^*_{\mathsf{lasso}} = \arg\min_{\boldsymbol{w}} \underbrace{\sum_{i=1}^{N}\left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j}w_j\right)^2}_{\mathsf{RSS}} + \lambda \underbrace{\sum_{j=1}^{d}|w_j|}_{\text{reg. function}} \tag{3.29}$$

$$= \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\tilde{\boldsymbol{w}}\|_1 \,. \tag{3.30}$$

Similar to ridge regression, the Lasso can be equivalently written as a constrained estimation problem as

$$\boldsymbol{w}^*_{\mathsf{lasso}} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N}\left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j}w_j\right)^2, \tag{3.31}$$

$$\text{subject to } \|\tilde{\boldsymbol{w}}\|_1 \leq t \tag{3.32}$$

which puts an explicit constraint on the size of parameters $\{w_j\}$.

The Lasso is similar to ridge regression with the key difference that the regularization function is now the ℓ_1 norm of $\tilde{\boldsymbol{w}}$. As a result, the solution of the Lasso is not a simple linear function of the y_i's as in ridge regression and, unlike ridge regression, there is no closed-form solution to the Lasso (because the term $\|\tilde{\boldsymbol{w}}\|_1$ is not differentiable whenever $w_j = 0$). Computing the Lasso solution is a quadratic programming problem (with a quadratic objective and linear constraints).

Just as for ridge regression, we can re-parametrize the constant w_0 by standardizing the predictors. The intercept can then be fitted separately (the solution is $\bar{\boldsymbol{y}}$, as for ridge regression) and then we can fit a model without an intercept to determine the other parameters. In the signal processing literature, the lasso is also known as basis pursuit.

The Lasso has the property that if λ is sufficiently large (equivalently t is sufficiently small), some of the coefficients w_j are driven to zero. Thus, the lasso can be interpreted as performing a sort of continuous subset selection.

3.7.1 Graphical interpretation of ridge regression and the lasso

In Figure 3.10, we give a graphical interpretation of ridge regression and the Lasso for two parameters w_1 and w_2. First, let's consider the least-squares regression. The RSS has elliptical contours, i.e., for a fixed RSS the solution to the least-squares regression problem is an ellipse, i.e., all pairs (w_1, w_2) corresponding the

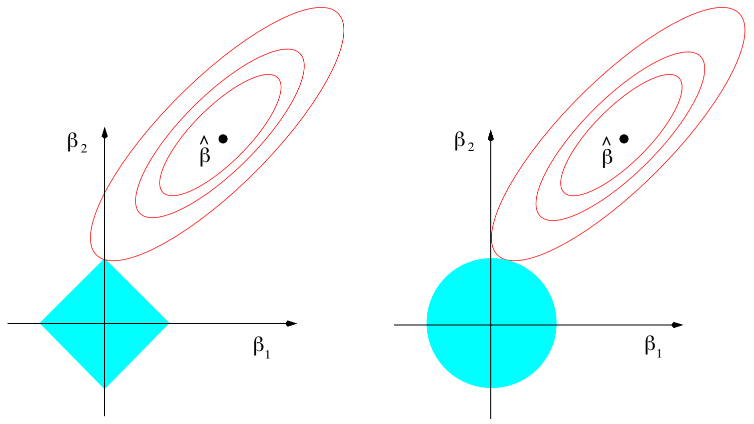

Figure 3.10: Visualization of estimate by the lasso (left) and ridge regression (right). The elliptical contours correspond to the pairs (w_1, w_2) that yield a given (fixed) RSS. The solid blue areas are the constrained regions $|w_1| + |w_2| \leq t$ and $w_1^2 + w_1^2 \leq t$, corresponding to the quadratic regularizer (ridge regression) and the ℓ_1 norm regularizer (lasso), respectively. (Figure from [1])

contour of one off these ellipses yield the same RSS. The elliptical contours are centered in the least-squares solution $\boldsymbol{w}^*$.

Now consider ridge regression. The constraint introduced by ridge regression, i.e., $\|\boldsymbol{w}\|^2 \leq t$ corresponds to the equation of a disk $w_1^2 + w_1^2 \leq t$ centered around zero (the feasible solutions for (w_1, w_2) lie within the disk). For the Lasso, the ℓ_1 term is the equation of a diamond centered around 0 (or a romboid in higher dimensions. A romboid has many vertices), $|w_1| + |w_2| \leq t$.

The solution to the constrained optimization lies at the intersection between the contours of the RSS and the constrained function (a disk or a diamond), and this intersection varies as a function of t (or equivelently λ). For $\lambda = 0$ the solution is the least-squares solution (as usual) and for $\lambda = \infty$ the solution is at $(0, 0)$.

Unlike the disk, the diamond has vertices. Note that at the vertices of the diamond, one of the parameters w_i is zero (in higher dimensions one or many of the variables have value 0). There is a non-zero probability that the intersection happens in one of the vertices, which implies that one or many of the coefficients will have a value exactly equal to 0. This does not occur for ridge regression.

Of course, the Lasso regression does't have to set coefficients to zero, in many cases it doesn't. What happens is that as you increase the λ parameter, the probability that the solution takes place at a vertex of the diamond increases, and so the probability that one or many coefficients is exactly zero also increases.

3.7.2 Generalization

We can of course generalize ridge regression and the lasso to a more general regularization of the least-squares as

$$\boldsymbol{w}^*_{\text{lasso}} = \arg\min_{\boldsymbol{w}} \underbrace{\sum_{i=1}^{N}\left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j} w_j\right)^2}_{\text{RSS}} + \lambda \underbrace{\sum_{j=1}^{d} |w_j|^q}_{\text{reg. function}} \tag{3.33}$$

The case $q = 1$ corresponds to the lasso, while $q = 2$ to ridge regression. Furthermore, $q = 0$ (ℓ_0 norm) corresponds to variable subset selection (the ℓ_0 norm corresponds to the total number of nonzero elements in a vector, in this case the number of nonzero parameters).

Regularization allows complex models to be trained on data sets of limited size without severe overfitting, essentially by limiting the effective model complexity. However, the problem of determining the optimal model complexity is then shifted from one of finding the appropriate number of basis functions to one of determining a suitable value of the regularization coefficient λ. We shall return to the issue of model complexity later.

Chapter 4

Linear methods for classification

We now turn to the classification problem. Particularly, we will discuss linear methods for classification. For classification, we recall that we are in a setting where the outcome $y \in \mathcal{G} = \{C_1, \dots, C_K\}$, i.e., it take values on a discrete set $\mathcal{G}$ of cardinality K. For convenience, we will encode the output $y = i$ if the class is C_i, i.e., $y \in \mathcal{G} = \{1, \dots, K\}$, except for the case of two classes, for which we will consider $y \in \mathcal{G} = \{0, 1\}$.

Given a training set $\mathcal{D}$ of N labeled examples $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, the classification problem is that to assign a new example $\boldsymbol{x}$ to one of the classes C_i. Classification therefore divides the input space into decision regions $\mathcal{R}_i$, $i \in [K]$, such that all points in $\mathcal{R}_i$ are assigned to class C_i. The boundaries of these decision regions are called decision boundaries.

In this chapter, we consider linear models for classification, by which we mean that the decision boundaries are linear functions of the input vector $\boldsymbol{x}$ and hence are defined by $(p-1)$-dimensional hyperplanes within the d-dimensional input space.

There are several approaches to tackle the classification problem. A popular one is to consider discriminative deterministic models, which model directly the deterministic mapping between each input $\boldsymbol{x}$ and class label via a parametrized function $\delta(\boldsymbol{x})$, called a discriminant function. A discriminant is a function that takes an input vector $\boldsymbol{x}$ and assigns it to one of the K classes. In this chapter, we shall restrict our attention to linear discriminants, namely those for which the decision boundaries are hyperplanes. For instance, in the case of two-class problems, $\delta(\boldsymbol{x})$ is binary valued and may be such that $\delta = 0$ represents class C_1 and $\delta = 1$ represents class C_2. We use the training data to find the discriminant function $\delta(\boldsymbol{x})$.

Consider first the case of two classes, which we have already discussed in, e.g., Section 2.2.1. There we discussed the option of coding the output as $y \in \{0, 1\}$, and performing least squares linear regression and using the decision rule (see (2.15))[1]

$$\hat{y}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } f(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|\boldsymbol{x}] = w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} \leq 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

so that we can interpret $f(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|\boldsymbol{x}] = w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}$ as the probability $p(y = 1|\boldsymbol{x})$.

Consider now the case of K classes, encoded as $1, \dots, K$. A common approach is to consider a single K-class discriminant comprising K linear functions of the form

$$f_i(\boldsymbol{x}) = w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x}$$

[1]For convenience, we will not carry over the superindex $*$ for the optimal parameter $\boldsymbol{w}$ all the time, but of course it should be understood that $\boldsymbol{w}$ is optimized.

and then assign a point $\boldsymbol{x}$ to class C_i if $f_i(\boldsymbol{x}) > f_j(\boldsymbol{x})$ for all $j \neq i$, i.e.,

$$\hat{y}(\boldsymbol{x}) = \operatorname*{argmax}_i f_i(\boldsymbol{x})\,.$$

Hence, the discriminant functions can be interpreted as the posterior probabilities $p(y = i|\boldsymbol{x})$ (we are building the discriminant functions so that they represent the probabilities $p(y = i|\boldsymbol{x})$). This can be seen as the generalization of the least squares linear regression in (4.1) for two classes to K classes.

The decision boundary between class C_i and class C_j is therefore given by $f_i(\boldsymbol{x}) = f_j(\boldsymbol{x})$ and hence corresponds to a $(p-1)$-dimensional hyperplane defined by $(\boldsymbol{w}_i - \boldsymbol{w}_j)^\mathsf{T}\boldsymbol{x} + (w_{i,0} - w_{j,0}) = 0$, i.e., the set of points $\{\boldsymbol{x} : (\boldsymbol{w}_i - \boldsymbol{w}_j)^\mathsf{T}\boldsymbol{x} + (w_{i,0} - w_{j,0}) = 0\}$. This has the same form as the decision boundary for the two-class case.

More generally, we can learn K discriminant functions $\delta_i(\boldsymbol{x}) = p(y = i|\mathbf{x} = \boldsymbol{x})$, one for each class C_i and make the decision as

$$\hat{y}(\boldsymbol{x}) = \operatorname*{argmax}_i \delta_i(\boldsymbol{x})\,.$$

This generates a linear decision boundary when there exists some monotone transformation g of $\delta_i(\boldsymbol{x})$ which is linear, i.e., g is a monotone function such that

$$g(\delta_i(\boldsymbol{x})) = w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x}\,.$$

In the following we discuss two discriminative methods: linear regression and logistic regression, which differ in the chosen discriminant functions $\delta_i(\boldsymbol{x})$. The first one falls within the class of discriminative deterministic models, while the second is a discriminative probabilistic model. Discriminative probabilistic models model the probability of a point $\boldsymbol{x}$ belonging to class C_i via a parametrized conditional pmf $p(\mathcal{C}_i|\boldsymbol{x})$.

4.1 Linear regression

Our problem is, given a training set $\mathcal{D}$ of N labeled examples $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, assign a new example $\boldsymbol{x}$ to one of the classes $C_1, \dots, C_K$.

The standard approach to applying linear regression for $K > 2$ is through one-hot encoding, in which now the outcome $\boldsymbol{y}$ is a vector of length K, $\boldsymbol{y} = (y_1, \dots, y_K)$, such that if the class is C_i then $y_i = 1$ and all other elements y_j of $\boldsymbol{y}$ are zero. We will then write $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, where each vector $\boldsymbol{y}_i$ is of length K, $\boldsymbol{y}_i = (y_{i,1}, \dots, y_{i,K})$. The idea is to decompose the problem into K binary problems of separating each class C_i from all the others, and coding each one of these as a linear regression problem.

Considering all N training examples, we can build the $N \times K$ matrix $\boldsymbol{Y}$, whose i-th row corresponds to the length-k output vector $\boldsymbol{y}_i$. Note that $\boldsymbol{Y}$ is a matrix in which each row has a single one (matrix $\boldsymbol{Y}$ is called in [1] an indicator response matrix).

In linear regression the discriminant functions $\delta_i(\boldsymbol{x})$, one for each column of $\boldsymbol{Y}$, are linear functions of $\boldsymbol{x}$, i.e.,

$$\delta_i(\boldsymbol{x}) = w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x}$$

where $\boldsymbol{w}_i = (w_{i,1}, \dots, w_{i,d})^\mathsf{T}$.

We have therefore a linear regression problem with multiple (K) outputs, which we have already addressed in Section 3.4. The least squares solution is given by (3.15),

$$\begin{aligned}\boldsymbol{B}^* &= \arg\min_{\boldsymbol{B}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{B}\|_{\mathsf{F}}^2 \\ &= (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{Y}\,.\end{aligned}$$

and the fit by

$$\hat{\boldsymbol{Y}}^* = \boldsymbol{X}\boldsymbol{B}^* = \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{Y}\,.$$

where $\boldsymbol{X}$ is the $N \times (p+1)$ data matrix and $\boldsymbol{B}$ the $(p+1)\times K$ matrix of coefficients

Each column of $\boldsymbol{B}^*$ is simply the least squares solution for one class, i.e., we have a coefficient vector $(w_{i,0}^*, \boldsymbol{w}_i^*)$, $i \in [K]$, for each column of $\boldsymbol{Y}$,

Hence, a new observation $\boldsymbol{x}$ is classified by

1. Computing the discriminant functions

$$\delta_i(\boldsymbol{x}) = w_{i,0}^* + (\boldsymbol{w}_i^*)^\top \boldsymbol{x}\,,$$

which are obtained via

$$\begin{aligned}\boldsymbol{\delta}(\boldsymbol{x}) = (\delta_i(\boldsymbol{x}), \dots, \delta_K(\boldsymbol{x}))^\top &= \big((1, \boldsymbol{x}^\top)\boldsymbol{B}^*\big)^\top \\ &= (\boldsymbol{B}^*)^\top (1, \boldsymbol{x}^\top)^\top\,.\end{aligned}$$

2. Select the class with corresponding maximum discriminant (maximum score),

$$\hat{y}(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, \delta_i(\boldsymbol{x})\,.$$

Note that if there are only two classes, all that matters is $\delta_1(\boldsymbol{x}) - \delta_2(\boldsymbol{x})$, so we really need only one function (see (4.1)).

If the matrix $\boldsymbol{X}$ contains an intercept, it is easy to see that $\sum_{i=1}^K \delta_i(\boldsymbol{x}) = 1$.

A major problem with linear regression for classification is the so-called *masking*—in the case where there is one class in the middle between two others, linear regression may not be able to predict the class in the middle. This occurs because of the rigid nature of the linear discriminant functions. An example is given in Figure 4.1 for $K = 3$ classes. The left plot shows the data for the three classes, which can be perfectly separated by linear decision boundaries. The classification arising from linear regression is shown in the right figure—linear regression misses most of the middle class completely! Of course this example is extreme, but for a large number of classes K and small d, such maskings occur naturally.

4.2 Logistic regression

Consider first the case of binary classification (i.e., two classes). Logistic regression considers the discriminant functions

$$\delta_i(\boldsymbol{x}) = p(\mathsf{y} = i|\mathbf{x} = \boldsymbol{x})\,,$$

with the class probabilities modeled as

$$p(\mathsf{y} = 0|\mathbf{x} = \boldsymbol{x}) = \frac{\exp(w_0 + \boldsymbol{w}^\top\boldsymbol{x})}{1 + \exp(w_0 + \boldsymbol{w}^\top\boldsymbol{x})} \tag{4.2}$$

$$= \frac{1}{1 + e^{-(w_0 + \boldsymbol{w}^\top\boldsymbol{x})}} \triangleq \sigma(w_0 + \boldsymbol{w}^\top\boldsymbol{x})\,, \tag{4.3}$$

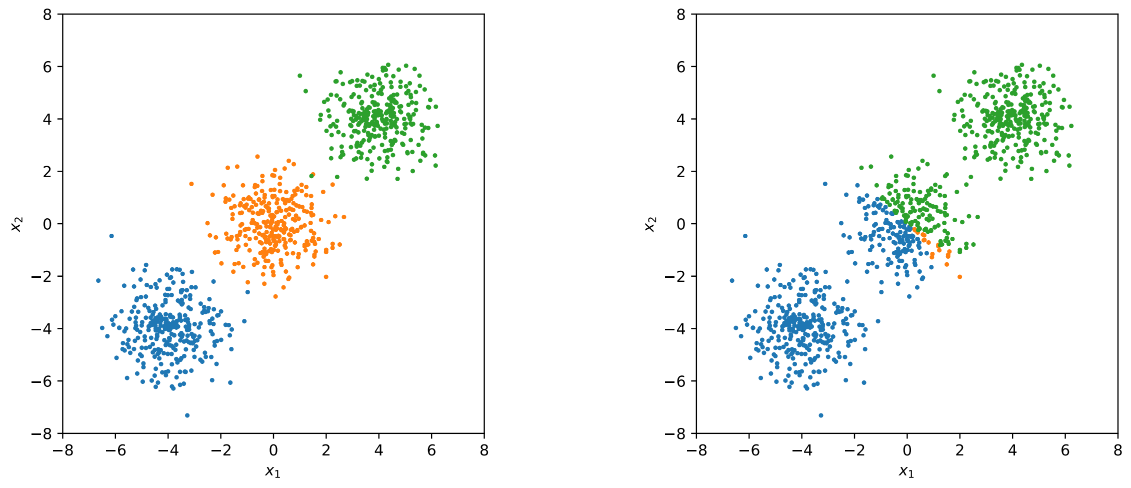

Figure 4.1: (Left) Real data, which can be easily separated into three classes by linear decision boundaries. (Right) Classification made by linear regression. As it can be seen, most of the middle class is missed.

where $\sigma(a) = \frac{1}{1+e^{-a}}$ is the sigmoid function, and

$$\begin{aligned} p(\mathrm{y}=1|\mathbf{x}=\boldsymbol{x}) &= 1 - p(\mathrm{y}=0|\mathbf{x}=\boldsymbol{x}) \\ &= \frac{1}{1+\exp(w_0+\boldsymbol{w}^\mathsf{T}\boldsymbol{x})}\,, \end{aligned} \tag{4.4}$$

such that applying the monotone transformation (called *logit* transformation or logit function)

$$g(p) = \log\left(\frac{p}{1-p}\right)$$

results in a linear function of the inputs,

$$g(p(y=0|\mathbf{x}=\boldsymbol{x})) = \log\frac{p(y=0|\mathbf{x}=\boldsymbol{x})}{p(y=1|\mathbf{x}=\boldsymbol{x})} \tag{4.5}$$

$$= w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}\,. \tag{4.6}$$

With logistic regression, the logit of the probability is a linear function of the inputs. Equation (4.5) represents the log of the ratio of probabilities for the two classes and is also sometimes called the "log odds" of class 0 versus class 1. As discussed, the classification for a new $\boldsymbol{x}$ is made again according to

$$\hat{y}(\boldsymbol{x}) = \operatorname*{argmax}_i \delta_i(\boldsymbol{x})\,,$$

which, using the log odds function can be equivalently written as (for class C_0) if

$$\hat{y}(x) = \begin{cases} 0 & \text{if } \log\frac{p(y=0|\mathbf{x}=\boldsymbol{x})}{p(y=1|\mathbf{x}=\boldsymbol{x})} \geq 0 \\ 1 & \text{if } \log\frac{p(y=0|\mathbf{x}=\boldsymbol{x})}{p(y=1|\mathbf{x}=\boldsymbol{x})} < 0 \end{cases}$$

i.e.,

$$\hat{y}(x) = \begin{cases} 0 & \text{if } w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} \geq 0 \\ 1 & \text{if } w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} < 0 \end{cases}$$

The decision boundary is therefore the set of points for which $w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}$ is zero, which defines the hyperplane $\{\boldsymbol{x} : w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = 0\}$.

For the case $K > 2$, logistic regression assumes the model

$$\log \frac{p(\mathsf{y} = i|\mathbf{x} = \boldsymbol{x})}{p(\mathsf{y} = K|\mathbf{x} = \boldsymbol{x})} = w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x} \tag{4.7}$$

for $i \in [K-1]$. It is important to remark that the use of class K as the reference class in the denominator is arbitrary—we could have used another (fixed) class.

We can easily transform this to probabilities for the classes, obtaining

$$p(\mathsf{y} = i|\mathbf{x} = \boldsymbol{x}) = \frac{\exp(w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x})}{1 + \sum_{j=1}^{K-1} \exp(w_{j,0} + \boldsymbol{w}_j^\mathsf{T}\boldsymbol{x})} \tag{4.8}$$

for $i \in [K-1]$, and

$$p(\mathsf{y} = K|\mathbf{x} = \boldsymbol{x}) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(w_{j,0} + \boldsymbol{w}_j^\mathsf{T}\boldsymbol{x})}\,.$$

which can be regarded as the multiclass generalization of logistic regression for two classes.

The posterior probabilities sum up to one.[2]

Note that this model induces linear decision boundaries between classes. Indeed, the decision boundary between class C_i and C_j is given by

$$\{\boldsymbol{x} \ : \ p(\mathsf{y} = i|\boldsymbol{x}) = p(\mathsf{y} = j|\boldsymbol{x})\}\,,$$

which, using (4.8) can be rewritten as

$$\{\boldsymbol{x} \ : \ (w_{i,0} - w_{j,0}) + (\boldsymbol{w}_i - \boldsymbol{w}_j)^\mathsf{T}\boldsymbol{x} = 0\}\,,$$

which describes a hyperplane.

4.2.1 Parameter optimization

To estimate the optimal parameters $\boldsymbol{B}$, given the training data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, one typically uses the ML criterion, maximizing the (log)-likelihood of the parameters. To simplify notation we define $\boldsymbol{\theta} = (w_{1,0}, \boldsymbol{w}_1^\mathsf{T}, w_{2,0}, \boldsymbol{w}_1^\mathsf{T}, \ldots, w_{K,0}, \boldsymbol{w}_K^\mathsf{T})$.[3]

Assuming that the training cases are independent, the likelihood function for the training data is just given by the product of the probabilities for each training point,

$$p(y_\mathcal{D}|x_\mathcal{D}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\,,$$

where we have made explicit the dependency on the parameters $\boldsymbol{\theta}$). The ML rule is then

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ p(y_\mathcal{D}|x_\mathcal{D}, \boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\,.\end{aligned}$$

[2]Note that this model assumes that $w_{K,i} = 0$ for all i.

[3]The book also defines $p(\mathsf{y} = i|\boldsymbol{x}) = p_i(\boldsymbol{x}; \boldsymbol{\theta})$. However, we will not use this notation, as it is an abuse of terminology, which makes unclear that the probability is a conditional probability.

Rather than maximizing the likelihood function, it is convenient to maximize the log of the likelihood function,

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ \log p(y_{\mathcal{D}}|x_{\mathcal{D}}, \boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\,.\end{aligned}$$

We will define $\ell(\boldsymbol{\theta}) \triangleq \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$.

We consider the two-class case, $\{C_0, C_1\}$. We will use the conventional coding $y_i = 1$ for class C_1 and $y_i = 0$ for class C_0. For the two-class case $\boldsymbol{\theta}$ simplifies to $\boldsymbol{\theta} = (w_0, \boldsymbol{w}^\mathsf{T})$, and we recall that the conditional probabilities for the two classes are given by (4.2) and (4.4).

Since either $y_i = 0$ or $1 - y_i = 0$ (at any time, only one of the two is nonzero), we can write $p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$ as

$$p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = p(\mathsf{y}_i = 1|\mathbf{x}_i = \boldsymbol{x}_i, \boldsymbol{\theta})^{y_i} p(\mathsf{y}_i = 0|\mathbf{x}_i = \boldsymbol{x}_i, \boldsymbol{\theta})^{1-y_i}\,,$$

and with this we obtain the log-likelihood $\log p(y_{\mathcal{D}}|x_{\mathcal{D}}, \boldsymbol{\theta})$ as

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log\big(p(\mathsf{y}_i = 1|\mathbf{x}_i = \boldsymbol{x}_i, \boldsymbol{\theta})^{y_i} p(\mathsf{y}_i = 0|\mathbf{x}_i = \boldsymbol{x}_i, \boldsymbol{\theta})^{1-y_i}\big) \\ &= \sum_{i=1}^{N} y_i \log p(\mathsf{y}_i = 1|\mathbf{x}_i = \boldsymbol{x}_i, \boldsymbol{\theta}) + (1 - y_i) \log p(\mathsf{y}_i = 0|\mathbf{x}_i = \boldsymbol{x}_i, \boldsymbol{\theta}) \\ &= \sum_{i=1}^{N} \Big[-y_i \boldsymbol{\theta}^\mathsf{T} \tilde{\boldsymbol{x}}_i - \log\Big(1 + e^{-\boldsymbol{\theta}^\mathsf{T} \tilde{\boldsymbol{x}}_i}\Big)\Big]\,, \qquad (4.9)\end{aligned}$$

where the last equality is obtained by defining $\tilde{\boldsymbol{x}}^\mathsf{T} = (1, \boldsymbol{x}^\mathsf{T})$, i.e., we include the constant term 1 in $\tilde{\boldsymbol{x}}$ to accommodate the intercept, $\boldsymbol{\theta}^\mathsf{T} = (1, \boldsymbol{w}^\mathsf{T})^\mathsf{T}$, and using (4.2) and (4.4).

Unfortunately, there is no simple formula to find the optimal $\boldsymbol{\theta}^*$ that maximizes (4.9). However, fortunately,

1. There is a unique maximum of $\ell(\boldsymbol{\theta})$ unless the classes are perfectly separated by a hyperplane,
2. The log-likelihood surface is convex,
3. The first and second derivatives of $\ell(\boldsymbol{\theta})$ can be easily computed.

Efficient methods like the Newton-Raphson iteration method can be used to find the point where $\nabla \ell(\boldsymbol{\theta}) = 0$, which is where the maximum occurs.

4.3 Generative probabilistic models

As we discussed in Section 2.4, statistical learning for classification requires the knowledge of the posterior probabilities $p(\mathsf{y} = C_i|\boldsymbol{x})$. Discriminative models model $p(\mathsf{y} = C_i|\boldsymbol{x})$ directly via discriminant functions.

Another approach to classification is generative probabilistic models, which model the joint distribution of the feature $\boldsymbol{x}$ and class label y by specifying the prior distribution $p(y)$, or $p(C_i)$, and the class-dependent probability distribution $p(\boldsymbol{x}|y)$, or $p(\boldsymbol{x}|C_i)$, of the domain points within each class.

Generative methods then obtain the posterior probabilities $p(\mathsf{y} = C_i|\boldsymbol{x})$ by applying the Bayes' theorem,

$$p(\mathsf{y} = C_i|\mathbf{x} = \boldsymbol{x}) = \frac{p(\mathbf{x}|\mathsf{y} = C_i)p(\mathsf{y} = C_i)}{p(\mathbf{x} = \boldsymbol{x})}\,. \qquad (4.10)$$

The book [1] defines $f_i(\boldsymbol{x})$ as the conditional probability $p(\boldsymbol{x}|\mathsf{y}=C_i)$, $f_i(\boldsymbol{x}) = p(\mathbf{x}|\mathsf{y}=C_i)$, and π_i as the prior probability of class C_i, i.e., $\pi_i = p(\mathsf{y}=C_i)$. However, here we will use $p(\boldsymbol{x}|C_i)$ and $p(\mathsf{y}=C_i)$ because it makes explicit the variables involved the two probability distributions.

We can rewrite (4.10) as

$$p(\mathsf{y}=C_i|\mathbf{x}=\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_i)p(\mathsf{y}=C_i)}{\sum_{j=1}^{K} p(\boldsymbol{x}|C_j)p(\mathsf{y}=C_j)}$$

Note that $p(\boldsymbol{x}|C_j)p(\mathsf{y}=C_j) = p(\boldsymbol{x},\mathsf{y}=C_j)$, and hence $\sum_{j=1}^{K} p(\boldsymbol{x},\mathsf{y}=C_j) = p(\boldsymbol{x})$.

To classify, a generative model requires a model for $p(\boldsymbol{x}|C_i)$ and knowledge or an estimate of the priors $p(\mathsf{y}=C_i)$.

Different models can be used for the class densities $p(\boldsymbol{x}|C_i)$, leading to different generative models:

- Linear discriminant analysis and quadratic discriminant analysis (QDA), which use Gaussian densities,
- Models using mixtures of Gaussians,
- General nonparametric density estimates,
- and Naive Bayes, where $p(\boldsymbol{x}|C_i) = \prod_{j=1}^{d} p(x_j|C_i)$.

4.3.1 Linear discriminant analysis

The key assumption of LDA is that the predictor vector $\boldsymbol{x} \in \mathbb{R}^d$ is normally distributed for each class C_i, with a common covariance matrix $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$ for all i, i.e., we model the conditional probabilities $p(\boldsymbol{x}|C_i)$ as a multivariate Gaussian,

$$\mathbf{x}|\mathsf{y}=C_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$$

or equivalently,

$$p(\boldsymbol{x}|C_i) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)}$$

Hence, fitting an LDA model to data requires estimating the common covariance matrix $\boldsymbol{\Sigma}$ and the class centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$.

LDA gives rise to linear decision boundaries. To see this, consider two classes i and j. We need to compare the probabilities $p(\mathsf{y}=C_i|\boldsymbol{x})$ and $p(\mathsf{y}=C_j|\boldsymbol{x})$ or equivalently, it is sufficient to consider the log ratio

$$\log \frac{p(\mathsf{y}=C_i|\boldsymbol{x})}{p(\mathsf{y}=C_j|\boldsymbol{x})}.$$

Expanding the log-ratio we obtain

$$\begin{aligned}
\log \frac{p(\mathsf{y}=C_i|\boldsymbol{x})}{p(\mathsf{y}=C_j|\boldsymbol{x})} &= \log \frac{p(\boldsymbol{x}|C_i)p(\mathsf{y}=C_i)}{p(\boldsymbol{x}|C_j)p(\mathsf{y}=C_j)} \\
&= \log \frac{p(\boldsymbol{x}|C_i)}{p(\boldsymbol{x}|C_j)} + \log \frac{p(\mathsf{y}=C_i)}{p(\mathsf{y}=C_j)} \\
&= -\frac{1}{2}\boldsymbol{\mu}_i^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \frac{1}{2}\boldsymbol{\mu}_j^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j + \boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \log \frac{p(\mathsf{y}=C_i)}{p(\mathsf{y}=C_j)} \qquad (4.11) \\
&= a_0 + \boldsymbol{a}^\mathsf{T}\boldsymbol{x},
\end{aligned}$$

which is a linear function.

The decision boundary between classes C_i and $\mathcal{C}_j$ is the set of points such that $\boldsymbol{x}^\mathsf{T}\boldsymbol{a} + b = 0$, which determines a hyperplane in d dimensions. This is of course true for any pair of classes, so all the decision boundaries are linear.

The optimal classification, under the maximum likelihood criterion, picks the class that maximizes the posterior probability given the feature vector $\boldsymbol{x}$,

$$\begin{aligned}
\hat{y}(\boldsymbol{x}) &= \underset{i}{\operatorname{argmax}}\, p(\mathsf{y} = i|\boldsymbol{x}) \\
&= \underset{i}{\operatorname{argmax}}\, \frac{p(\mathbf{x}|\mathsf{y} = C_i)p(\mathsf{y} = C_i)}{p(\mathbf{x} = \boldsymbol{x})} \\
&= \underset{i}{\operatorname{argmax}}\, p(\mathbf{x}|\mathsf{y} = i)p(\mathsf{y} = C_i) \\
&= \underset{i}{\operatorname{argmax}}\, \log(p(\mathbf{x}|\mathsf{y} = i)p(\mathsf{y} = C_i)) \\
&= \underset{i}{\operatorname{argmax}}\, -\log\Big((2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}\Big) - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) + \log p(\mathsf{y} = C_i) \\
&= \underset{i}{\operatorname{argmax}}\, -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) + \log p(\mathsf{y} = C_i) \\
&= \underset{i}{\operatorname{argmax}}\, -\frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_i^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \log p(\mathsf{y} = C_i) \\
&= \underset{i}{\operatorname{argmax}}\, \boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + -\frac{1}{2}\boldsymbol{\mu}_i^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \log p(\mathsf{y} = C_i)\,.
\end{aligned} \tag{4.12}$$

This is the final classifier. Given any $\boldsymbol{x}$, we simply plug it into this formula and seek which i maximizes this expression. From this, we can define the linear discriminant functions

$$\delta_i(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\boldsymbol{\mu}_i^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \log p(\mathsf{y} = C_i)\,,$$

and the decision rule is

$$\hat{y}(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, \delta_i(\boldsymbol{x})\,.$$

The decision boundary between class i and j is

$$\{\boldsymbol{x} \ : \ \delta_i(\boldsymbol{x}) = \delta_j(\boldsymbol{x})\}\,.$$

To perform the classification task, we need the parameters of the Gaussian distributions $\{\boldsymbol{\mu}_i\}$, $\boldsymbol{\Sigma}$, and $\{p(\mathsf{y} = C_i)\}$. Obviously, in practice, they are unknown, and hence we need to estimate them. We can do using the training data. Let n_i is the number of class C_i observations, $n_i = \sum_{j=1}^N \mathbb{1}\{y_j = i\}$, $\hat{p}(\mathsf{y} = C_i)$ the estimation of $p(\mathsf{y} = C_i)$, $\hat{\boldsymbol{\mu}}_i$ the estimation of the mean vector $\boldsymbol{\mu}_i$ for class C_i, and $\hat{\boldsymbol{\Sigma}}$ the estimation of $\boldsymbol{\Sigma}$. Then, the maximum likelihood estimator of $p(\mathsf{y} = C_i)$ is

$$\hat{p}(\mathsf{y} = C_i) = \frac{n_i}{N}\,,$$

where n_i is the number of class C_i observations. To get the prior probabilities for class i, we simply count the frequency of data points in this class.

Estimating the mean vector for every class is also simple. We simply take all of the data points in a given class and compute their average, i.e., the sample mean,

$$\hat{\boldsymbol{\mu}}_i = \frac{\sum_{j:y_j=i}\boldsymbol{x}_j}{n_i}\,,$$

Finally, the covariance matrix is estimated as

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N-K} \sum_{j=1}^{K} \sum_{i:y_i=j} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j)^{\mathsf{T}} \,.$$

As an interesting property, if $K = 2$ and $p(\mathsf{y} = C_1) = p(\mathsf{y} = C_2)$, then the optimal decision boundary between classes is the same for LDA and least squares regression classification (with $(0, 1)$ coding).

4.3.2 Quadratic discriminant analysis

For quadratic discriminant analysis, we assume

$$\mathsf{x}|\mathsf{y} = C_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i),$$

or equivalently,

$$p(\boldsymbol{x}|C_i) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}_i|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^{\mathsf{T}}\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)} \,,$$

allowing a different covariance matrix for each class. The effect is that the decision boundaries between classes are no longer linear.

We should classify a new input $\boldsymbol{x}$ as a class C_i is $p(\boldsymbol{x}|C_i)p(\mathsf{y} = C_i)$ is greater for this i than for any other. Proceeding as for LDA (expanding the log ratios) we see that corresponds to using the discriminant functions

$$\delta_i(\boldsymbol{x}) = -\frac{1}{2}\log|\boldsymbol{\Sigma}_i| - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}}\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) + \log p(\mathsf{y} = C_i)$$

Note that δ_i is a quadratic function of $\boldsymbol{x}$, so the decision boundary between each pair of classes is given by a quadratic equation, $\{\boldsymbol{x} : \delta_i(\boldsymbol{x}) = \delta_j(\boldsymbol{x})\}$, which gives name to QDA.

In Figure 4.2(left) we apply QDA to a classification problem with three classes (in blue, orange, and green) defined by three different Gaussian mixtures. The colored numbers correspond to training data points, where the color indicates the class they belong. Particularly, the data in each class is generated according to a given Gaussian mixture. The purple curves are the decision boundaries obtained by QDA. Observe that they are quadratic.

There are two alternative ways to realize quadratic decision boundaries. The first is obviously to run QDA in the original d-dimensional space. An alternative is to run LDA with an expanded variable set of dimension p^2 which includes quadratic terms and cross products, $x_1, \ldots, x_p, x_1^2, \ldots, x_p^2, x_1x_2, \ldots, x_{p-1}x_p$. Linear functions in the augmented space correspond to quadratic functions in the original d-dimensional space. Figure 4.2(right), shows the decision boundaries obtained by LDA for the same data as in Figure 4.2(left), but where LDA is not applied directly to the data, but to the enlarged five-dimensional quadratic polynomial space x_1, x_2, x_1x_2, x_1^2 and x_2^2. Note that linear functions in this space translate into quadratic functions in the original space. Applying LDA directly on the original space, i.e., two-dimensional features $\boldsymbol{x} = (x_1, x_2)$ would have given rise to linear decision boundaries.

The two methods do not yield the same boundaries, but the differences are small, as is usually the case. Note also that the LDA option does not really make sense with the probabilistic assumptions, since if the x_i's are Gaussian distributed, then x_i^2 and x_ix_j are not Gaussian by definition.

Both LDA and QDA perform very well on a large variety of classification tasks and are widely used in practice. This might be a bit surprising, perhaps, as these methods model $p(\boldsymbol{x}|C_i)$ as a Gaussian distribution, i.e., they make the implicit assumption that the data are approximately Gaussian (and in addition for LDA

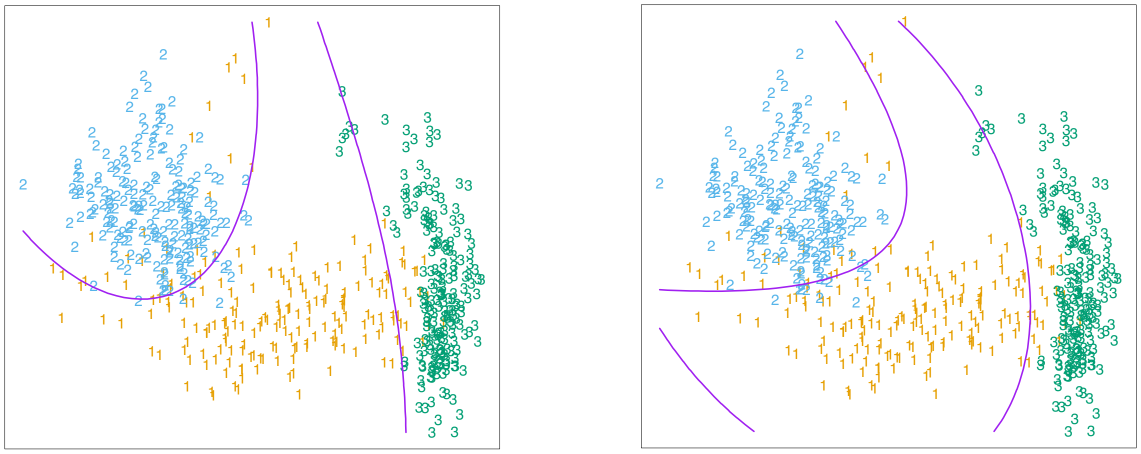

Figure 4.2: Three-class classification problem via via QDA and LDA. (Left) Decision boundaries obtained applying QDA. (Right) Decision boundaries obtained applying LDA in the five-dimensional space $x_1, x_2, x_1x_2, x_1^2, x_2^2$. (Figure from [1])

that the covariances are approximately equal), which is not necessarily the case in practice. A reason for the good performance of LDA and QDA may be that the data can only support simple decision boundaries such as linear or quadratic and, furthermore, the estimates provided by Gaussian models are stable.

4.3.3 Regularized discriminant analysis

One problem of QDA is that it substantially increases the number of parameters, due to the different covariance matrices for each class. As a result, the model is of course richer, which will translate into a smaller bias. However, it will potentially feature a larger variance. This justifies applying regularization to trade off bias and variance. In regularized discriminant analysis the covariance matrices have the form

$$\boldsymbol{\Sigma}_i(\alpha) = \alpha\boldsymbol{\Sigma}_i + (1-\alpha)\boldsymbol{\Sigma}\,,$$

which combines the global estimate of the covariance (the one used in LDA) with the class-specific ones. This offers interesting trade-offs between bias and variance or between linearity and non-linearity. $\alpha = 0$ corresponds to LDA and $\alpha = 1$ to QDA.

4.4 Connection between LDA and logistic regression

It is interesting to consider the resemblance between LDA and logistic regression. Consider the log-odds of LDA (see (4.11))

$$\begin{aligned}\log\frac{p(\mathsf{y}=i|\boldsymbol{x})}{p(\mathsf{y}=K|\boldsymbol{x})} &= -\frac{1}{2}\boldsymbol{\mu}_i^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \frac{1}{2}\boldsymbol{\mu}_K^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_K + \boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_K) + \log\frac{p(\mathsf{y}=C_i)}{p(\mathsf{y}=C_K)}\\ &= a_{i,0} + \boldsymbol{a}_i^\mathsf{T}\boldsymbol{x}\,,\end{aligned}$$

which is a linear function of $\boldsymbol{x}$. This linearity results from the Gaussian model for the class densities and the equal covariance matrix for all classes.

For logistic regression,

$$\log\frac{p(\mathsf{y}=i|\boldsymbol{x})}{p(\mathsf{y}=K|\boldsymbol{x})} = w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x}\,.$$

The two models have therefore the same form. Indeed, the model of LDA satisfies the assumption of the linear logistic model (linear log-odds).

However, the way the coefficients are estimated for logistic regression and LDA are not equivalent. LDA makes stronger assumptions than logistic regression, and therefore the latter is more general. Particularly, the linear logistic model only specifies the conditional distribution $p(\mathsf{y} = i|\mathbf{x} = \boldsymbol{x})$. Indeed, both LDA and logistic regression assume the conditional distribution of the form

$$p(\mathsf{y} = i|\mathbf{x} = \boldsymbol{x}) = \frac{\exp(w_{i,0} + \boldsymbol{w}_i^\mathsf{T}\boldsymbol{x})}{1 + \sum_{j=1}^{K-1}\exp(w_{j,0} + \boldsymbol{w}_j^\mathsf{T}\boldsymbol{x})}\,.$$

Logistic regression makes no assumption about $p(\boldsymbol{x})$, while LDA specifies the joint distribution $p(y, \boldsymbol{x})$,

$$\begin{aligned} p(\mathsf{y} = i, \mathbf{x} = \boldsymbol{x}) &= p(\mathbf{x}|\mathsf{y} = i)p(\mathsf{y} = i) \\ &= p(\mathbf{x}|\mathsf{y} = i; \boldsymbol{\mu}_i, \boldsymbol{\Sigma})p(\mathsf{y} = C_i)\,, \end{aligned}$$

where $p(\mathbf{x}|\mathsf{y} = i; \boldsymbol{\mu}_i, \boldsymbol{\Sigma})$ is a multivariate Gaussian, from which it results that $p(\boldsymbol{x})$ is a mixture of Gaussians,

$$p(\boldsymbol{x}) = \sum_{i=1}^{K} p(\boldsymbol{x}|\mathsf{y} = i; \boldsymbol{\mu}_i, \boldsymbol{\Sigma})p(\mathsf{y} = C_i)\,.$$

Moreover, logistic regression is solved by maximizing the conditional likelihood of y given $\boldsymbol{x}$, $p(y|\mathbf{x} = \boldsymbol{x})$, while LDA maximizes the joint likelihood $p(y, \boldsymbol{x})$ (see (4.12)). Note that logistic regressions models $p(y|\mathbf{x} = \boldsymbol{x})$ directly, while LDA models $p(\boldsymbol{x}|y)$ and from this $p(y|\mathbf{x} = \boldsymbol{x})$.

If the additional assumptions made by LDA are appropriate, LDA tends to estimate the parameters more efficiently by using more information about the data. On the other hand, because logistic regression relies on fewer assumptions, it seems to be more robust to the non-Gaussian type of data.

In practice, logistic regression and LDA often give similar results.

4.5 Separating hyperplanes

Consider the binary classification of the data set in $\mathbb{R}^2$ in Figure 4.3, which shows two classes (blue and red circles) that are separable by a hyperplane. The orange line shows the least squares solution, which coincides with the solution given by LDA. Despite the fact that the two classes can be separated by a hyperplane, least squares linear regression and LDA are not able to completely separate the two classes. In this section, we will consider separating hyperplane classifiers, which determine linear decision boundaries (hyperplanes) that explicitly separate the data into different classes.

We focus again on binary classification. The training set for a binary classification problem is linearly separable if there is a hyperplane in the input space that separates training examples of one class from those of the other class. If the examples are linearly separable, there will be an infinite number of hyperplanes that separate them.

With d feeatures, the hyperplane will be of dimension $p - 1$, and consists of the points $\boldsymbol{x}$ that satisfy the equation

$$\{x \,:\, w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = 0\}. \tag{4.13}$$

A hyperplane is a line when $p = 2$, a plane when $p = 3$, and, more generally, a $p - 1$-dimensional affine subspace in the domain space.

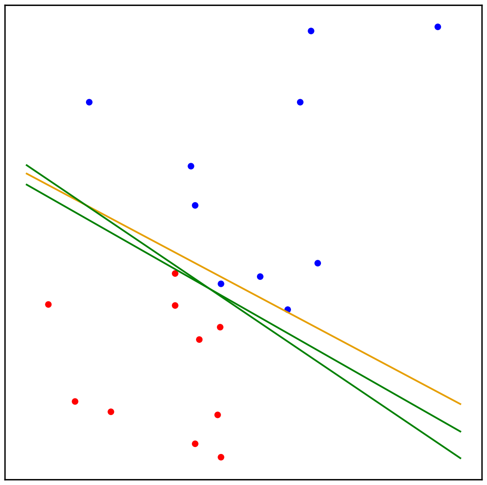

Figure 4.3: Two classes separable by a hyperplane. The orange line corresponds to the least squares solution, which misclassifies one of the training points. The two green lines correspond to separating hyperplanes found by the perceptron algorithm with different initializations of $\tilde{\boldsymbol{w}}$.

In what follows, it is more convenient to encode the two classes C_0 and C_1 by -1 and 1, respectively, rather than by 0 and 1, i.e., we assume $y \in \{-1, 1\}$.

Points on one side of the hyperplane will satisfy $w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} > 0$, and points on the other side $w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} < 0$. We will use the convention that the side > 0 corresponds to a response $y = 1$, and the side < 0 to $y = -1$. Accordingly, in their simplest form, linear discriminative deterministic classification models are of the form

$$\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname{sign}(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) \,, \tag{4.14}$$

where the sign function returns 1 if its argument is positive, and -1 if the argument is negative.

Let's consider the geometric interpretation of the decision rule (4.14) for the case of two dimensions. This decision rule defines a hyperplane, in this case a line, that separates the domain points classified as belonging to either of the two classes. The decision hyperplane can be identified as described in Figure 4.4. Consider two points $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ that lie in the decision hyperplane, i.e., satisfy (4.13). Since $y(\boldsymbol{x}_a) = y(\boldsymbol{x}_b) = 0$, it follows that $\boldsymbol{w}^\mathsf{T}(\boldsymbol{x}_a - \boldsymbol{x}_b) = 0$, and hence the vector $\boldsymbol{w}$ defines the direction perpendicular to the hyperplane, i.e., $\boldsymbol{w}$ is normal to the hyperplane and, consequently, determines the orientation of the decision hyperplane. Also, if $\boldsymbol{x}$ is a point on the decision surface, then $y(\boldsymbol{x}) = 0$, i.e., $w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = 0$. The distance from a point (x_0, y_0) to the line $ax + by + c = 0$—defined as the shortest distance between a fixed point and any point on the line, i.e., the length of the line segment that is perpendicular to the line and passes through the point—is

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

Thus, the distance from the origin to the decision surface is given by

$$\frac{|w_0|}{\|\boldsymbol{w}\|} \,,$$

and the signed distance is

$$\frac{-w_0}{\|\boldsymbol{w}\|} \,.$$

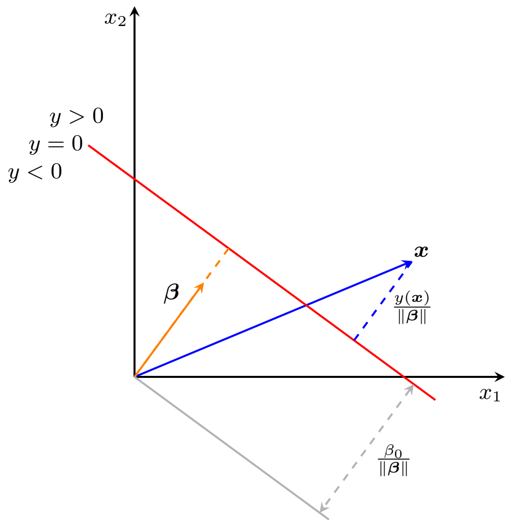
x_2
$y > 0$
$y = 0$
$y < 0$
$\boldsymbol{x}$
$\boldsymbol{\beta}$
$\frac{y(\boldsymbol{x})}{\|\boldsymbol{\beta}\|}$
x_1
$\frac{\beta_0}{\|\boldsymbol{\beta}\|}$

Figure 4.4: Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\boldsymbol{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\boldsymbol{x}$ from the decision surface is given by $y(\boldsymbol{x})/\|\boldsymbol{w}\|$.

Therefore, the parameter w_0 determines the location of the decision hyperplane.

We may be interested in how far a point $\boldsymbol{x}$ is from the hyperplane. We note that the value of $y(\boldsymbol{x})$ gives a signed measure of the perpendicular distance a of the point $\boldsymbol{x}$ from the decision surface. To see this, consider an arbitrary $\boldsymbol{x}$ and its orthogonal projection $\boldsymbol{x}_\perp$ onto the decision hyperplane. We can write

$$\boldsymbol{x} = \boldsymbol{x}_\perp + a\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

where $\|\boldsymbol{w}\|$ is the length of $\boldsymbol{w}$, hence $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ is a vector of length one, and a is the length of the orthogonal projection of $\boldsymbol{x}$.

Multiplying both sides by $\boldsymbol{w}^\mathsf{T}$ and adding w_0 we obtain,

$$w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_\perp + a\boldsymbol{w}^\mathsf{T}\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\,,$$

which we can rewrite as

$$y(\boldsymbol{x}) = y(\boldsymbol{x}_\perp) + a\|\boldsymbol{w}\|\,.$$

as $\boldsymbol{x}_\perp$ lies in the hyperplane, $y(\boldsymbol{x}_\perp) = 0$ and we obtain

$$a = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|} = \frac{w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}}{\|\boldsymbol{w}\|}\,.$$

The absolute (unsigned) distance between $\boldsymbol{x}$ and the decision hyperplane is then given by

$$\frac{|w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}|}{\|\boldsymbol{w}\|}\,.$$

This distance can be understood as a measure of the confidence level at which the classifier assigns $\boldsymbol{x}$ to the class identified via the rule (4.14).

Suppose that the training data is linearly separable. How should we find a separating hyperplane? Note that, although all separating hyperplanes classify all the training cases examplees, they might perform very differently when classifying test—yet unseen—data. Obviously, the goal is to choose a separating hyperplane that will do well on test cases, i.e., that it will generalize well. In the following we discuss an algorithm that attempts this.

4.5.1 The perceptron algorithm

The perceptron algorithm, introduced by Frank Rosenblatt in 1957 for binary classification, is one of the very first machine learning algorithms. It tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.

The perceptron algorithm considers the loss function (called the perceptron loss function)

$$\ell(y, \tilde{\boldsymbol{w}}) = \max(0, -y(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}))\,,$$

where it is assumed that the output y takes values in $\{-1, 1\}$, $y \in \{-1, 1\}$. We will define $\tilde{\boldsymbol{w}} = \left(w_0, \boldsymbol{w}^\mathsf{T}\right)^\mathsf{T}$.

The perceptron algorithm assigns zero cost to a correctly classified example $\boldsymbol{x}$, and a cost equal to the absolute value of $y(w_0 + \boldsymbol{w}^\mathsf{T})$ for a misclassified example. Accordingly, the output $y = 1$ is misclassified if $(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) < 0$, and $y = -1$ is misclassified if $(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) > 0$. In a more compact form, a case with input $\boldsymbol{x}$ and response y is correctly classified if

$$y(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) \geq 0\,,$$

and incorrectly classified if

$$y(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) < 0\,.$$

The perceptron algorithm chooses $\tilde{\boldsymbol{w}}^*$ that minimizes

$$\begin{aligned}\tilde{\boldsymbol{w}}^* &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} \ell(y_i, \tilde{\boldsymbol{w}}) \\ &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} \max(0, -y_i(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i))\,. \qquad (4.15)\end{aligned}$$

The quantity $\sum_{i=1}^{N} \max(0, -y_i(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i))$ is nonnegative and proportional to the distance of the misclassified points to the decision boundary defined by $w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = 0$.

The perceptron algorithm solves (4.15) via stochastic gradient descent with mini-batch size $s = 1$. The resulting algorithm is given in the following.

Algorithm (The perceptron algorithm)

Initialization: Initialize $\tilde{\boldsymbol{w}}^{(0)}$.

For each iteration $j = 1, 2 \ldots$,

1. Pick a training example $(\boldsymbol{x}_i, y_i)$ uniformly with replacement from $\mathcal{D}$;
2. If the example is correctly classified, i.e., $y(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) \geq 0$, do not update the weights, i.e.,

$$\tilde{\boldsymbol{w}}^{(j)} \leftarrow \tilde{\boldsymbol{w}}^{(j-1)}\,.$$

Otherwise, if the example is not correctly classified, i.e., $y(w_0 + \boldsymbol{w}^\mathsf{T}\boldsymbol{x}) < 0$, update the weights as

$$\tilde{\boldsymbol{w}}^{(i)} \leftarrow \tilde{\boldsymbol{w}}^{(j-1)} - \rho \nabla_{\tilde{\boldsymbol{w}}} \ell(y_i, \tilde{\boldsymbol{w}})\big|_{\tilde{\boldsymbol{w}}=\tilde{\boldsymbol{w}}^{(j-1)}} = \tilde{\boldsymbol{w}}^{(j-1)} + \rho y_i (1, \boldsymbol{x}_i^\mathsf{T})^\mathsf{T} \,,$$

where ρ is the learning rate. (As the perceptron function $y(\boldsymbol{x}, \tilde{\boldsymbol{w}})$ is unchanged if we multiply $\tilde{\boldsymbol{w}}$ by a constant, we can set the learning rate parameter ρ equal to 1 without of generality.)

3. Repeat the steps above until no points are misclassified.

It can be shown that, if the training set is linearly separable, the perceptron algorithm converges to a separating hyperplane in a finite number of steps. However, it should be noted that convergence can be slow. More importantly, the perceptron fails on training sets that are not linearly separable—the algorithm will never converge. This realization came as a disappointment and contributed to the first so-called AI winter period characterized by a reduced funding for AI and machine learning research.

Note also that when the data is separable there may be many solutions (hyperplanes that separate classes); which one the algorithm finds, depends on the initialization of the parameters, $\tilde{\boldsymbol{w}}^{(0)}$, and on the order data points are selected. All separating hyperplanes are considered equally valid, but they may perform differently on unknown data. In Figure 4.3, for the same data set considered earlier, we plot two separating hyperplanes (in green) found by the perceptron algorithm with two different initializations for $\tilde{\boldsymbol{w}}^{(0)}$.

Chapter 5

Moving beyond linearity

In the previous chapters, we have discussed linear models for regression and classification. For example, for linear regression, we assume a model

$$f(\boldsymbol{x}) = w_0 + w_1 x_1 + \ldots, w_p \boldsymbol{x}_p = \boldsymbol{x}^\mathsf{T} \boldsymbol{w} \,,$$

which is a linear function of the parameters $w_1, \ldots, w_p$ and also of the input variables $x_1, \ldots, x_p$. This imposes significant limitations on the model as in practice, the true function $\tilde{f}(\boldsymbol{x})$ will most likely be nonlinear with respect to the input variables. In this chapter, we introduce the concept of basis expansion (which we briefly discussed in Chapter 2) and one of the most important nonlinear regression techniques: regression splines.

The basic idea of basis expansion is to extend the linear regression model by considering linear combinations of nonlinear functions of the input variables,

$$f(\boldsymbol{x}) = w_0 + \sum_{i=1}^{M} w_i \phi_i(\boldsymbol{x}) \,,$$

where the functions $\phi_i(\boldsymbol{x})$ are known as basis functions or basis expansion.

The nice property of this approach is that the resulting model is still a linear function of the parameters $w_1, \ldots, w_p$ and of the basis functions $\phi(\boldsymbol{x})$. Indeed, these models are often called linear models as well, because they are linear in $\boldsymbol{w}$.

Note that if $\phi_i(\boldsymbol{x}) = x_i$ (and $M = p$), we recover the linear regression model.

By using nonlinear basis functions, we allow the function $f(\boldsymbol{x})$ to be a nonlinear function of the feature vector $\boldsymbol{x}$, which allows to model more complicated regression relationships or decision boundaries (in the case of classification). The number of basis functions M determines the capacity of the model. The larger M, the more capable the model will be able to fit the data. However, a larger M may lead to overfitting. To avoid overfitting we may resort to, e.g., regularization, as for ridge regression and Lasso.

Another particular instance of the basis expansion model is polynomial regression, in which, for $p = 1$, $\phi(x)$ is of the form[1]

$$\phi_i(x) = x^i \,,$$

so that we model $f(x)$ as

$$f(x) = \sum_{i=0}^{M} w_i x^i \,.$$

[1]In the case where $\boldsymbol{x}$ is a vector, the exponentiation is pointwise, $\phi(\boldsymbol{x}) = (1, \boldsymbol{x}, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^k)^\mathsf{T}$.

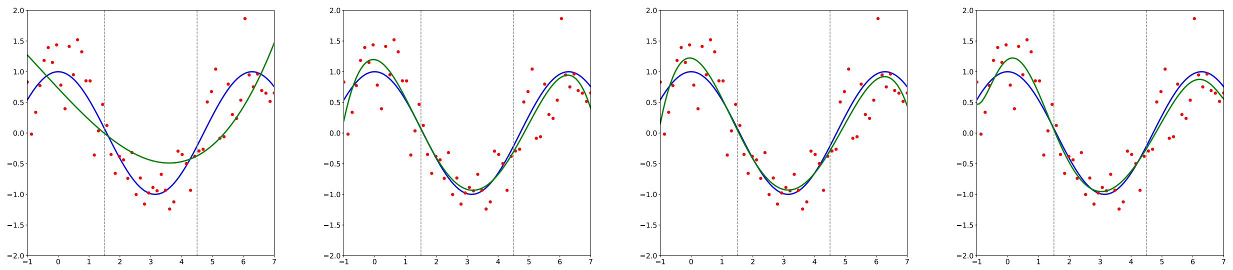

Figure 5.1: Illustration of polynomial regression. The blue curve corresponds to the true function $\cos\left(\frac{\pi}{3}x\right)$ and the red circles to the data generated from (5.1). (left) Degree-3 polynomial. (second from the leftt) Degree-4 polynomial. (second from the right) Degree-5 polynomial. (right) Degree-8 polynomial.

5.1 Piecewise polynomials and splines

Piecewise polynomials and splines are a particular class of linear basis function models that allow to capture local behavior in the data—they allow for local polynomial representation of the data.

We will assume for simplicity and exposition that the data is one-dimensional, i.e., $\boldsymbol{x} = x$ and as usual, we assume that the data arises from the probabilistic model

$$y = \tilde{f}(x) + \varepsilon$$

with $\mathbb{E}[\varepsilon] = 0$. We want to find a function $f(x)$ which is a good approximation to the true conditional expectation $\mathbb{E}[\mathsf{y}|x] = \tilde{f}(x)$ based on a given data set. To do so, we could consider polynomial interpolation by considering one polynomial of degree M for the whole domain of the data.

In Figure 5.1, we consider polynomial regression to fit the data generated from the probabilistic model

$$\cos\left(\frac{\pi}{3}x\right) + \varepsilon\,, \tag{5.1}$$

where $\varepsilon \sim \mathcal{N}(0, 0.16)$. The blue curve corresponds to the true function $\cos\left(\frac{\pi}{3}x\right)$ and the red circles to the data generated from (5.1), and the green curve to the learned polynomial. Particularly, we consider polynomials of degree 3 (left), degree 4 (second from the left), degree 5 (second from the right) and degree 8 (right), for which the vector of parameters $\boldsymbol{w}$ are optimized based on the least-squares criterion.

Increasing the degree polynomial allows to fit better the data (increasing the degree too much, however, would lead to overfitting). Polynomial regression, however, imposes a global structure on the data set and hence may not capture well local behavior. This could be problematic for data for which behavior is different in different regions. To overcome this and track better local behavior, a very popular approach is to use piecewise polynomial functions and splines, which we discuss next.

To obtain a piecewise function $f(x)$, we divide the domain of x, $x \in [a, b]$, into contiguous intervals and fit a polynomial function on each of these intervals. The points where the division of the domain of x occurs are called *knots*, and the functions that model the data in each interval are known as piecewise functions. There are several piecewise functions that we can use to fit these individual intervals. For example, we could use piecewise step functions, which are functions that remain constant within an interval. In this case, if we divide the domain of x into N_{I} intervals, $f(x)$ can be written as

$$f(x) = \sum_{i=1}^{N_{\mathsf{I}}} w_i \phi_i(x)$$

with

$$\begin{aligned}\phi_1(x) &= \mathbb{1}\{x < \xi_1\}\\ \phi_2(x) &= \mathbb{1}\{\xi_1 \le x < \xi_2\}\\ &\vdots\\ \phi_{N_\mathsf{I}}(x) &= \mathbb{1}\{\xi_{N_\mathsf{I}-1} \le x\}\end{aligned}$$

We then fit the model $f(x) = \sum_{i=1}^{N_\mathsf{I}} w_i\phi_i(x)$ (i.e., we find the parameter $\boldsymbol{w}^*$) to the data using least-squares. As the basis functions do not overlap, we fit a different function $f_i(x)$ for each interval $i \in [N_\mathsf{I}]$. We have

$$f_i(x) = w_i$$

and the optimal w_i^* resulting from least-squares regression is the mean of the y_i's in this interval,

$$w_i^* = \arg\min_{w_i} \sum_{j=1}^{N} (y_j - w_i)^2 .$$

Taking the derivative with respect to w_i and equating to zero,

$$2\sum_{j=1}^{N}(y_j - w_i) = 0 \quad\Longleftrightarrow\quad w_i^* = \frac{1}{N}\sum_{j=1}^{N} y_j .$$

In Figure 5.2(left), we consider piecewise step functions to fit the data generated from the function (5.1). We divide the domain $[a, b]$ into three intervals delimited by ξ_1 and ξ_2. $[a, \xi_1)$, $[\xi_1, \xi_2)$, and $[\xi_2, b]$, where ξ_1, ξ_1, and ξ_1 are the knots, and we fit to each interval a constant function. We can write $f(\boldsymbol{x})$ as

$$f(x) = \sum_{i=1}^{3} w_i\phi_i(x)$$

with the three basis functions

$$\phi_1(x) = \mathbb{1}\{x < \xi_1\}, \qquad \phi_2(x) = \mathbb{1}\{\xi_1 \le x < \xi_2\}, \qquad \phi_3(x) = \mathbb{1}\{\xi_2 \le x\}$$

We then fit the model using least-squares, obtaining $w_i^* = \frac{1}{N}\sum_{i=1}^{N} y_i$. The fitted piecewise constant functions are shown in green.

Instead of fitting a constant function for each interval, piecewise polynomial regression involves fitting separate (low-degree) polynomials to the different regions of x. Consider first a piecewise linear function, i.e., polynomials of degree one. The piecewise linear function can be defined as

$$f(x) = \sum_{i=1}^{6} w_i\phi_i(x) \tag{5.2}$$

with

$$\begin{aligned}&\phi_1(x) = \mathbb{1}\{x < \xi_1\}, && \phi_2(x) = \mathbb{1}\{\xi_1 \le x < \xi_2\}, && \phi_3(x) = \mathbb{1}\{\xi_2 \le x\}\\ &\phi_4(x) = x\phi_1(x) && \phi_5(x) = x\phi_2(x), && \phi_6(x) = x\phi_3(x)\end{aligned}$$

Again, the basis functions do not overlap, hence we can fit a separate linear model to the data in each region using least-squares. Indeed, for $f(x)$ defined in (5.2), we obtain a linear function for each of the three

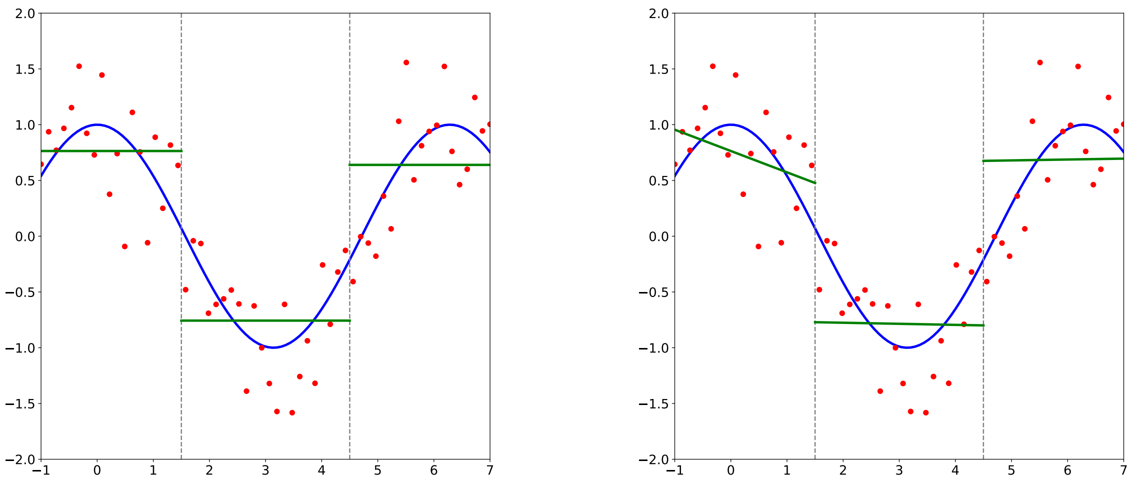
Figure 5.2: Illustration of the use of piecewise polynomials to fit data. The blue curve corresponds to the true function $\cos\left(\frac{\pi}{3}x\right)$ and the red circles to the data generated from (5.1). (left) Piecewise constant function. (right) Piecewise linear function.

intervals: $f_1(x) = w_1 + w_4x$, $f_2(x) = w_2 + w_5x$, and $f_3(x) = w_3 + w_6x$, respectively. In other words, we fit three different linear functions to the data. The optimal w_i^*'s are found using least-squares.

To achieve a smoother $f(\boldsymbol{x})$, potentially fitting better the data, we can increase the order of the polynomials, leading to piecewise polynomial functions. For example, a piecewise quadratic polynomial function is obtained by fitting a quadratic polynomial

$$f_i(x) = w_{i,0} + w_{i,1}x + w_{i,2}x^2$$

to each of the intervals, and a piecewise cubic polynomial function by fitting a polynomial of degree three,

$$f_i(x) = w_{i,0} + w_{i,1}x + w_{i,2}x^2 + w_{i,3}x^3$$

In Figure 5.3(left), we plot the result of fitting the data using a piecewise cubic polynomial function. Obviously, by increasing the degree of the polynomial, we achieve a better fit.

Another way to improve the fit is obviously to use more knots, which leads to a more flexible piecewise polynomial—we use a different polynomial for each interval.

Looking at Figures (5.2) and (5.3)(left), we see that we need to be careful while using piecewise polynomials. For example, in the figures we observe a large discontinuity between the different functions at the knots, resulting from the fact that we have fitted the polynomials independently. This should be clearly avoided, as we would like the family of polynomials as a whole to generate a unique output for every input, which does not occur now at the knots—at each knot, the two functions at either end of the knot generate a different output. To avoid this, we can add the extra constraint that the fitted curve is continuous piecewise, i.e., the polynomials on either side of a knot should evaluate to the same value the knot. For the piecewise linear function for the example above, introducing the constraint that the linear functions for two contiguous regions are identical at the knot can be done by setting the constraints

$$w_1 + w_4\xi_1 = w_2 + w_5\xi_1, \qquad \text{and} \qquad w_2 + w_5\xi_2 = w_3 + w_6\xi_2\,.$$

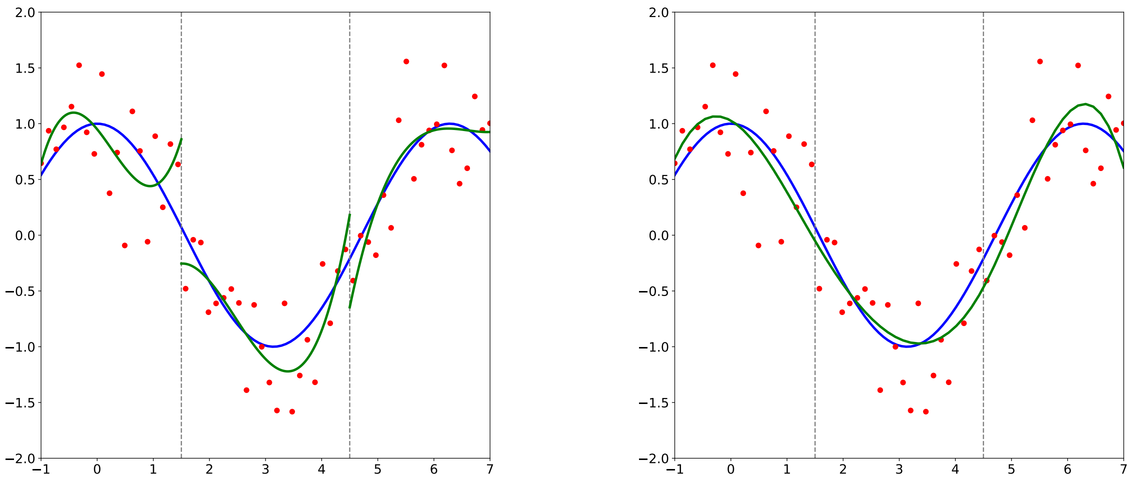
Figure 5.3: Illustration of the use of a cubic polynomials and a cubic spline to fit data. The blue curve corresponds to the true function $\cos\left(\frac{\pi}{3}x\right)$ and the red circles to the data generated from (5.1). (left) Piecewise cubic polynomial function. (right) Cubic spline.

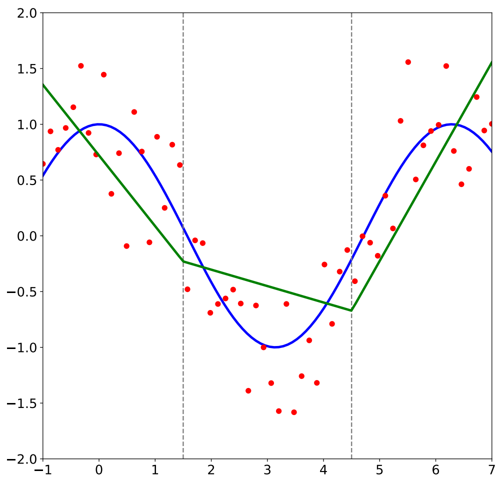
Figure 5.4: Piecewise linear function avoiding discontinuities at the knots.

Alternatively, we can use basis functions that incorporate the contraints directly,

$$f(x) = \sum_{i=1}^{4} w_i \phi_i(x)$$

with

$$\phi_1(x) = 1, \qquad \phi_2(x) = x, \qquad \phi_3(x) = (x - \xi_1)_+, \qquad \phi_4(x) = (x - \xi_2)_+\,,$$

where a_+ denotes the positive part of a, i.e., $a_+ = \max(a, 0)$. The continuous piecewise linear function obtained by imposing these constraints is show in Figure 5.4.

Consider now the cubic polynomials. In this case, after adding the constraint that the prediction at a knot should be the same for polynomials at either side of the knot, we obtain a function with no discontinuities, but not smooth. The way to solve this is to require that also both the first and second derivative are continuous (they are evaluated to the same value at the knots), i.e., we impose the constraints

$$f_i(\xi_i) = f_{i+1}(\xi_i), \qquad f'_i(\xi_i) = f'_{i+1}(\xi_i), \qquad f''_i(\xi_i) = f''_{i+1}(\xi_i) \tag{5.3}$$

for $i \in [K]$, where K is the number of knots. This yields a smooth line fitting the data, as shown in Figure 5.3(right). Such a piecewise polynomial of degree 3 with 2 continuous derivatives is called a *cubic spline*, i.e., a cubic spline is a piecewise cubic function that is continuous, and has continuous first and second derivatives. Incorporating the constraints in the basis functions, we can express a cubic spline by

$$f(x) = \sum_{i=1}^{6} w_i \phi_i(x)$$

with

$$\begin{aligned} &\phi_1(x) = 1, && \phi_2(x) = x, && \phi_3(x) = x^2 \\ &\phi_4(x) = x^3 && \phi_5(x) = (x-\xi_1)_+^3, && \phi_6(x) = (x-\xi_2)_+^3 . \end{aligned}$$

More generally, an order-M spline is a piecewise polynomial function of degree $M-1$ that is continuous and has continuous derivatives of orders $1, \ldots, M-2$ at its knot points. Formally, a function $f : \mathbb{R} \to \mathbb{R}$ is an M-th order spline with knot points at $\xi_i, \ldots, \xi_K$ if

- f is a polynomial of degree $M-1$ in each of the intervals $(-\infty, \xi_1], [\boldsymbol{x}_1, \xi_2), \ldots [\boldsymbol{x}_K, \infty)$, and
- The j-th derivative of f is continuous at knot points $\xi_i, \ldots, \xi_K$ for each $j = 0, \ldots, M-2$.

An order-M spline with K knots at points $\boldsymbol{x}_1, \ldots, \xi_K$ is a parametric model that can be parametrized by using the so-called truncated power basis functions

$$\begin{aligned} \phi_i(x) &= x^{i-1}, && i = 1, \ldots, M \\ \phi_{M+j}(x) &= (x-\xi_j)_+^{M-1}, && j = 1, \ldots, K . \end{aligned}$$

which are obtained from the polynomials

$$f_i(x) = w_{i,0} + w_{i,1}x + w_{i,M-1}x^{M-1}$$

with constraints (5.3). The number of degrees of freedom is thus $M+K$. Cubic splines are the most-widely used.

Note that each constraint that we impose on the piecewise polynomials reduces the number of degrees of freedom (free parameters) of the resulting function—we reduce the complexity of the resulting piecewise polynomial fit. For the piecewise cubic polynomial example, since each polynomial has four parameters, with three knots, we have a total of twelve degrees of freedom to fit this piecewise function. Each constraint reduces by 2 the number of degrees of freedom (one for each knot). Thus, imposing continuity up to the second derivative reduces by 6 the number of degrees of freedom to 6.

To design a spline, we need to select the order of the spline (typically we will choose a cubic spline), the number of knots and the position of these knots. A common approach is to set a knot at each observation x_i. If we consider a fixed number of knots (i.e., regardless how many training examples we have), it is it is common to place them in a uniform fashion. Splines with a fixed set of K knots are known as *regression splines*. Regression splines is one of the most important nonlinear regression techniques for smoothly interpolating data.

The coefficients $w_1, \ldots, w_{M+K}$ of the spline are just estimated by least squares, i.e., we find $w_1^*, \ldots, w_{M+K}^*$ to minimize

$$\sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{M+K} w_j \phi_j(x_i) \right)^2 . \tag{5.4}$$

Define the $N \times (K+M)$ basis matrix $\mathbf{\Phi}$ with entries

$$\phi_{i,j} = \phi_j(x_i), \quad i = 1, \ldots, N, \; j = 1, \ldots, K+M \,,$$

i.e., the j-th column $\mathbf{\Phi}$ gives the evaluations of ϕ_j over the points $x_1, \ldots, x_N$. Let $\boldsymbol{\phi}_i = (\phi_{i,1}, \ldots \phi_{i,M+K})^\mathsf{T}$, i.e., the i-th row of $\mathbf{\Phi}$. With this, we can rewrite (5.4) as

$$\sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{M+K} w_j \phi_{i,j} \right)^2 = \sum_{i=1}^{N} \left(y_i - \boldsymbol{\phi}_i^\mathsf{T} \boldsymbol{w} \right)^2 \tag{5.5}$$

$$= \| \boldsymbol{y} - \mathbf{\Phi} \boldsymbol{w} \|^2 \,, \tag{5.6}$$

which is the standard form for least squares regression. Note that now the vectors of features $\boldsymbol{\phi}_i$ is form by applying a transformation to the original data x via the basis functions $\phi_j(x_i)$.

From what we know about linear regression, the optimal coefficients are (see (2.14))

$$\boldsymbol{w}^* = (\mathbf{\Phi}^\mathsf{T} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\mathsf{T} \boldsymbol{y} \,,$$

and the fitted values are

$$\begin{aligned} \hat{\boldsymbol{y}} &= \mathbf{\Phi} \boldsymbol{w}^* \\ &= \mathbf{\Phi} (\mathbf{\Phi}^\mathsf{T} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\mathsf{T} \boldsymbol{y} \,. \end{aligned}$$

5.2 Natural cubic splines

One problem with polynomial regression is that their estimates tend to display erratic behavior—i.e., they may have high variance—at the boundaries of the domain of the data. This problem is inherited by splines. A way to remedy this problem and smooth the spline beyond the boundary knots, is to force the piecewise polynomials to have a lower degree to the left of the leftmost knot and to the right of the rightmost knot. This is what a particular type of spline, known as natural spline, does: A natural spline adds the additional constraint that the function is linear beyond the boundary knots, i.e., in the region left to the leftmost knot and the region right to the rightmost one. The effect of the additional constraints are generally more stable estimates at the boundaries. This constrains the cubic and quadratic parts there to be zero ($f''(x) = f'''(x) = 0$), each reducing the degrees of freedom by 1. In total the number of degrees of freedom is reduced by 4. Hence, a natural cubic spline with K knots can be represented by $M + K - 4 = 4 + K - 4 = K$ basis functions.

The fact that the model for natural cubic splines is linear on the boundaries, prevents it from extrapolating too wildly beyond the range of the data.

5.3 Smoothing splines

Regression splines work well provided we choose well the location of the knots $\xi_1, \ldots, \xi_K$. In general, however, choosing knots is tricky. Smoothing splines are a class of splines which do not require to select the knots.

They circumvent the problem of knot selection by using the inputs as knots, i.e., we place one knot at each data input. The basic idea of smoothing splines is to fit a function $f(x)$ by minimizing its the discrepancy to the data plus a smoothing term that penalizes functions that are too wiggly.

Consider all functions $f(x)$ with two continuous derivatives. A way to penalize too wiggly functions is to penalize the second derivative of the function, i.e., we want to find the function $f(x)$ that minimizes the objective function

$$\mathsf{RSS}(f,\lambda) = \sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda \underbrace{\int (f''(t))^2 \mathrm{d}t}_{\text{curvature penalty}} , \tag{5.7}$$

where λ is a regularization parameter, often called the smoothing parameter. The first term measures the discrepancy of $f(x)$ to the data—it is just the mean squared error of using the curve $f(x)$ to predict y. The second term penalizes wiggling functions. $f''(x)$ is the second derivative of $f(x)$, which is zero if $f(x)$ is linear, hence this measures the curvature of $f(x)$ at x. We then integrate this over all x to say how curved $f(x)$ is on average.

Parameter λ it establishes a trade-off between predicting the training data and minimizing the curvature of $f(x)$. Given two functions with the same least-squares error, we prefer the one with less average curvature. Note that $\lambda = \infty$ corresponds to the least squares line fit, while for $\lambda = 0$ we don't introduce any restriction to the function $f(x)$.

Interestingly, on can show that the solution of (5.7) is a natural cubic spline with knots at the data points $x_1, \ldots, x_N$. Hence, with respect to the criterion in (5.4) for regression splines, we have an extra term, the regularization term. Thus, smoothing splines perform a regularized regression over the natural spline basis, placing knots at all data points. Furthermore, they simultaneously control for overfitting by shrinking the coefficients of the estimated function (in its basis expansion).

We can write $f(x)$ as

$$f(x) = \sum_{j=1}^{N} w_j \phi_j(x) ,$$

where $\phi_1(x), \ldots, \phi_N(x)$ are the N basis functions of a natural spline with N knots.

As before, by defining the $N \times N$ basis matrix $\boldsymbol{\Phi}$ with entries

$$\phi_{i,j} = \phi_j(x_i), \quad i,j = 1,\ldots,N .$$

we can write the first term in (5.7) as $\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|^2$ (see (5.6)). Further, by defining the penalty matrix $\boldsymbol{\Omega}$ with entries

$$\Omega_{i,j} = \int \phi_i''(t)\phi_j''(t), \quad i,j = 1,\ldots,N .$$

we can write the second term in (5.7) as

$$\lambda \int (f''(t))^2 \mathrm{d}t = \lambda \boldsymbol{w}^\mathsf{T}\boldsymbol{\Omega}\boldsymbol{w} .$$

Thus,

$$\begin{aligned} \mathsf{RSS}(f,\lambda) &= \sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda \int (f''(t))^2 \mathrm{d}t && (5.8)\\ &= \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|^2 + \lambda \boldsymbol{w}^\mathsf{T}\boldsymbol{\Omega}\boldsymbol{w} . && (5.9) \end{aligned}$$

Similar to least squares regression, the optimal coefficients $\boldsymbol{w}^*$, obtained by minimizing (5.9), are

$$\boldsymbol{w}^* = (\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \lambda\boldsymbol{\Omega})^{-1}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}\,,$$

and the fitted values are

$$\begin{aligned}\hat{\boldsymbol{y}} &= \boldsymbol{\Phi}\boldsymbol{w}^* \\ &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \lambda\boldsymbol{\Omega})^{-1}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}\,.\end{aligned}$$

Note that (5.8) can be seen as a generalized version of ridge regression (see (3.16)).

Remark 5.1 Smoothing splines often deliver similar fits to those from kernel regression. Both have a tuning parameter—the bandwidth for kernel regression, and the smoothing parameter λ for smoothing splines—, which we need to choose (typically by cross-validation as discussed in the next chapter. However, splines are in a sense simpler, as we don't require a choice of kernel. Furthermore, smoothing splines are generally much more computationally efficient.

Chapter 6

Model assessment and selection

In this chapter, we address the following questions: how can we assess the performance of a given learning algorithm, and how do we select a good model? We have already discussed that one would like to minimize the generalization error, which is related to the capability of the learning algorithm to predict the outcome of new, previously unseen data. This assessment will guide the choice of the learning algorithm.

Consider first the regression problem, i.e., we have a target variable y to estimate from a vector of inputs $\boldsymbol{x}$, and we learn a prediction model $f(\boldsymbol{x})$ from the training data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$. Further, we measure the cost incurred by the estimate $\hat{y}$ via a loss function $\ell(y, f(\boldsymbol{x}))$, for example a quadratic loss $\ell(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$.

Assuming a fixed training set, the test error (or generalization error) over an independent test sample is given by (see (2.2) in Section 2.1)

$$\begin{aligned}\mathsf{Err}(\mathcal{D}) &= \mathbb{E}[\ell(\mathsf{y}, f(\mathbf{x}))|\mathcal{D}] \\ &= \mathbb{E}_{\mathbf{x},\mathsf{y}}[\ell(\mathsf{y}, f(\mathbf{x}))|\mathcal{D}]\,,\end{aligned} \tag{6.1}$$

where $\mathbf{x}$ and y are drawn from the (unknown) joint distribution $p(\boldsymbol{x}, y)$.

Considering all randomness, the expected prediction error is

$$\begin{aligned}\mathsf{Err} &= \mathbb{E}_{\mathbf{x},\mathsf{y},\mathcal{D}}[\ell(\mathsf{y}, f(\mathbf{x}))] \\ &= \mathbb{E}_{\mathcal{D}}[\mathsf{Err}(\mathcal{D})]\,,\end{aligned} \tag{6.2}$$

where now the expectation is taken over all possible training sets $\mathcal{D}$, i.e., over all random quantities y, $\mathbf{x}$, and $\mathcal{D}$ (different training sets $\mathcal{D}$ will lead to different functions $f(\boldsymbol{x})$, as the function is learned from the training set). In general, we would like to estimate $\mathsf{Err}(\mathcal{D})$.

We can also define the training error (or training loss), i.e., the average loss over the training set, which is given by

$$L_{\mathcal{D}} = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, f(\boldsymbol{x}_i))\,,$$

where $\boldsymbol{x}_i \in \mathcal{D}$.

Figure 6.1 shows the generalization error (6.1) (light red curves) for 100 different training sets, the expected prediction error (6.2) (dark red curve), and the training error (light blue curves), together with its average over the 100 training sets (dark blue curve) as a function of the model complexity. The considered algorithm

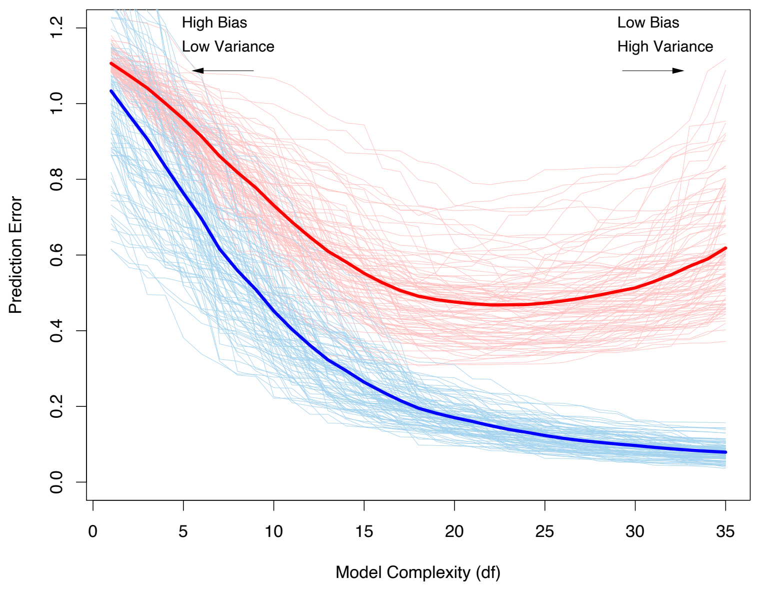

Figure 6.1: Training error and prediction error as a function of the model complexity. The light blue curves correspond to the training error for 100 different training sets, and the dark blue curve to the average. The light red curves correspond to the generalization error for 100 different training sets, and the dark red curve to the average prediction error. (Figure from [1])

is the lasso, for which we recall that the complexity is determined by parameter λ. The smaller λ is (the less regularization), the higher the complexity of the model.

For low model complexity, the training error is large. In this case, the model $f(\boldsymbol{x})$ underfits the data, i.e., it is not rich enough to capture all the variations in the data. The training error decreases with model complexity: the richer the model, the better it will be able to fit the data. As expected, however, the training error is not a good estimate of the generalization error. Particularly, while for high model complexity the training error is small (and will tend to zero), the generalization error initially decreases with model complexity but eventually it will increase. Indeed, a very rich model will tend to overfit the data and will typically generalize poorly—the model essentially memorizes the training set, rather than learning how to generalize to previously unseen data. The same principle applies to classification, in which case the loss function is different, e.g., the 0–1 loss function

$$\ell(y, f(\boldsymbol{x})) = \mathbb{1}\{y \neq f(\boldsymbol{x})\}\,.$$

As we discussed, for both regression and classification, the model $f(\boldsymbol{x})$ is typically a parametric model, parametrized by a set of tunable parameters $\boldsymbol{\theta}$. Examples of parametric models are polynomial regression, for which the tunable parameters are the degree of the polynomial and the vector $\boldsymbol{w}$, or the regularized least squares (e.g., ridge regression or the lasso), for which the parameters are the regularization parameter λ, which controls the effective complexity of the model, and $\boldsymbol{w}$. To make this explicit, we will sometimes write $f(\boldsymbol{x}; \boldsymbol{\theta})$.

6.1 Hypothesis class and model parameters

In order to learn an approximation of $\tilde{f}(\boldsymbol{x})$, we first need to select a family of parametric models, known as the hypothesis class, and then, given the hypothesis class, learn the specific parameters of the model to minimize the generalization error. Examples of a hypothesis class are ridge regression, the lasso, or smoothing splines, with a specific value of λ. Different values of λ determine different hypothesis classes, and control the complexity of the model. For ridge regression, determining a specific model within a given hypothesis class (i.e., for a given λ), requires the selection of the parameters $\boldsymbol{w}$. Another example is polynomial regression, for which the polynomial degree M defines the hypothesis class, and a specific model within this class is obtained by selecting $\boldsymbol{w}$. Parameters that define the hypothesis class, such as λ and M, are known as *hyperparameters*. Note that in model selection and validation, we usually use *model* to refer to a particular hypothesis class for learning. This shouldn't be confused with the specific model (with all parameters trained) to be used.

Both hyperparameters and regular parameters play a significantly different role during learning and should be clearly distinguished, as discussed next.

- The hyperparameters define the capacity of the hypothesis class to fit the available data. For example, in ridge regression and the lasso, a larger λ reduces the effective capacity of the model.
- Given a hypothesis class, assigning specific values to the model parameters (e.g., $\boldsymbol{w}$) identifies a specific model within the hypothesis class.

The learning and assessment of a model comprises the following three steps:

- **Hypothesis class selection**: We consider a set of different hypothesis classes and we select the best one for our particular application (in a way that will be defined precisely later).
- **Specific model selection**: Within the selected hypothesis class, we select a specific model, i.e., the particular parameters (e.g., $\boldsymbol{w}$ in case of ridge regression).
- **Model assessment**: For the final chosen instance of the model, estimate its generalization error on new data.

The first two items correspond to model selection.

We would like to proceed as follows. We will consider ridge regression as an example. In this case, we will start by considering several values of λ, i.e., several hypothesis classes, $\lambda \in \mathcal{L} = \{\lambda_1, \ldots, \lambda_{\mathsf{r}}\}$. Then, for each hypothesis class, we fit the parameters $\boldsymbol{w}$ using the training set. Among the resulting specific models (one for each value of λ) we would like to choose the one that minimizes the generalization error. This, however, assumes that the generalization error $\mathsf{Err}(\mathcal{D})$ (and hence the expected prediction error as well) can somehow be evaluated. However, these depend on the true, unknown, distribution $p(\boldsymbol{x}, y)$, and thus strictly speaking this evaluation is not possible. How can we then estimate the generalization error to perform model selection and assessment? The conventional procedure is the so-called *validation*. We can proceed as follows: We divide the available data into three sets: a training set $\mathcal{D}$, a validation set $\mathcal{V}$ —also called hold-out set—, and a test set $\mathcal{T}$, as shown in Figure 6.2. We denote by N, N_{v}, and N_{t}, the number of examples in the training set, validation set, and test set, respectively.

The training set is used to fit the models. Then, the validation set is used to evaluate an approximation of the generalization error via the empirical average

$$L_{\mathcal{V}} = \frac{1}{N_{\mathsf{v}}} \sum_{i=1}^{N_{\mathsf{v}}} \ell(y_i, f(\boldsymbol{x}_i))$$

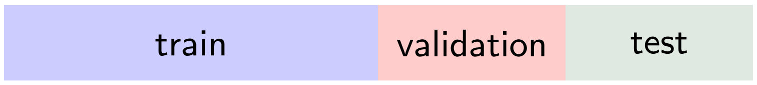

Figure 6.2: For training and validation, data may be split into three sets.

to enable model selection. Particularly, we select the model that minimizes the generalization error on the validation set. Finally, once the model is selected via learning and validation, we need to produce an estimate of the generalization error obtained with this choice. The generalization error estimated via validation cannot be used for this purpose, as it tends to be smaller than the actual value of the generalization error—we have selected the model to yield the smallest possible error on the validation set! Hence, the assessment of the generalization error should be done on separate data, which is referred to as the test set. We expect that the generalization error computed on the test set will be a good estimate of the generalization error on new, previously unseen data. A typical split of the data is 50% for training, 25% for validation, and 25% for testing, or 60%, 20%, and 20%.

In the following, we describe more precisely how validation is performed via a particular example.

Example 6.1 (Training and validation for ridge regression)

We consider training and validation assuming different hypothesis classes within ridge regression. We proceed as follows.

1. The first step is to choose a set of regularization parameters λ, $\lambda \in \mathcal{L} = \{\lambda_1, \ldots, \lambda_{\mathsf{r}}\}$, i.e., a set of hypothesis classes.

2. Then, we proceed to select the best hypothesis class—hyperparameter λ. We proceed as follows. For each $\lambda \in \mathcal{L}$, we learn the optimal $\boldsymbol{w}$ as

$$\boldsymbol{w}^*_\lambda = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} \left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j} w_j \right)^2 + \lambda \sum_{j=1}^{d} w_j^2 \,,$$

   where the learning is performed using the training set $\mathcal{D}$.

   We then select λ that minimizes

$$L_{\mathcal{V}}(f(\boldsymbol{x}; \lambda, \boldsymbol{w}^*_\lambda)) = \frac{1}{N_{\mathsf{v}}} \sum_{i=1}^{N_{\mathsf{v}}} \ell(y_i, f(\boldsymbol{x}_i; \lambda, \boldsymbol{w}^*_\lambda)) = \frac{1}{N_{\mathsf{v}}} \sum_{i=1}^{N_{\mathsf{v}}} \left(y_i - w^*_{\lambda,0} - \sum_{i=1}^{d} x_{i,j} w^*_{\lambda,j} \right)^2 ,$$

   i.e.,

$$\lambda^* = \arg\min_{\lambda \in \{\mathcal{L}\}} L_{\mathcal{V}}(f(\boldsymbol{x}; \lambda, \boldsymbol{w}^*_\lambda)) \,.$$

3. We have now chosen the hypothesis class. Then, for selected hypothesis class, i.e., the selected λ, which we denote by λ^*, we retrain $\boldsymbol{w}$ based on the data in the training and validation set, i.e., $\mathcal{D} \cup \mathcal{V}$, to select the specific model within the selected hypothesis class.

4. Finally, for chosen specific model, given by parameters $(\lambda^*, \boldsymbol{w}^*_{\lambda^*})$, we estimate the generalization error based on the test set $\mathcal{T}$ (which is newly seen data, as it has not been used for training and

validation),

$$L_{\mathcal{T}}(f(\boldsymbol{x};\lambda^*,\boldsymbol{w}^*_{\lambda^*})) = \frac{1}{N_{\mathsf{t}}}\sum_{i=1}^{N_{\mathsf{t}}}\ell(y_i, f(\boldsymbol{x}_i;\lambda^*,\boldsymbol{w}^*_{\lambda^*}))\,.$$

We expect this to give a good estimate of the generalization error.

The validation approach described above suffers from a clear drawback: part of the available data needs to be set aside and not used for training, i.e., the number of examples that can be used for training is smaller than the number of overall available data points. Therefore, this approach is suitable in situations in which we have plenty of data available. In many applications, however, the available data for training, validation, and testing will be limited. Obviously, we would like to use as much of the available data for training, yielding a more accurate model. However, if the validation set is small, it will provide a noisy estimate of the performance. Thus, in these situations dividing the data into a training, validation, and test set is not suitable and we need to proceed differently. There are several methods to perform validation in situations where there is insufficient data available. Next, we describe the most widely-used method, called k-fold cross-validation.

6.2 k-fold cross-validation

With this method, the available data points are randomly partitioned into k equally-sized partitions, called folds. The generalization error is then estimated by averaging k different estimates. Particularly, for each of the k partitions, $\kappa \in \{1,\dots,k\}$, a model is trained using the other $k-1$ partition as training data and the resulting model is validated on the κ-th partition by computing the corresponding generalization error on this partition. The generalization error estimated by k-fold cross-validation is then the average of the k values for the generalization error thus computed.

As for validation, the primary purpose of cross-validation is not to provide the final specific model, but model selection, i.e., hypothesis class selection. Assume that we consider two hypothesis classes, for example ridge regression with two different values of λ. We can use k-fold cross-validation to see which hypothesis class (which λ) proves better at predicting the test set points. Once we have selected the best hypothesis class, we then need to find the best specific model within that class. For this, we train the chosen model (ridge regression with a specific λ in this case) on all the data.

Consider a set of models $f(\boldsymbol{x};\boldsymbol{\alpha},\boldsymbol{\theta})$ indexed by the hyperparameters $\boldsymbol{\alpha}$ (e.g., $\boldsymbol{\alpha}=\lambda$ for ridge regression) and regular parameters $\boldsymbol{\theta}$ (e.g., $\boldsymbol{w}$). Further, let $\kappa \in \{1,\dots,k\}$ be the κ-th partition of the data, each containing N_κ examples, and $L_{\mathcal{V}}(f(\boldsymbol{x};\boldsymbol{\alpha},\kappa,\boldsymbol{\theta}))$ the generalization error of the model for this partition, obtained by training the model on the other $k-1$ partitions (i.e., using a total of $(k-1)N_\kappa$ data points) and evaluating the generalization error over the κ-th partition. $L_{\mathcal{V}}(f(\boldsymbol{x};\boldsymbol{\alpha},\kappa,\boldsymbol{\theta}))$ is computed as

$$L_{\mathcal{V}}(f(\boldsymbol{x};\boldsymbol{\alpha},\kappa,\boldsymbol{\theta})) = \frac{1}{N_\kappa}\sum_{i=1}^{N_\kappa}\ell(y_i, f(\boldsymbol{x}_i;\boldsymbol{\alpha},\boldsymbol{\theta}))\,,$$

where the sum runs over the N_κ training points in the κ-th partition.

The k-fold cross-validation estimate of the generalization error for a hypothesis class (or model) $f(\boldsymbol{x};\boldsymbol{\alpha})$ is then

$$L_{k-\mathsf{fold}}(f(\boldsymbol{x};\boldsymbol{\alpha})) = \frac{1}{k}\sum_{\kappa=1}^{k}L_{\mathcal{V}}(f(\boldsymbol{x};\boldsymbol{\alpha},\kappa,\boldsymbol{\theta}))\,.$$

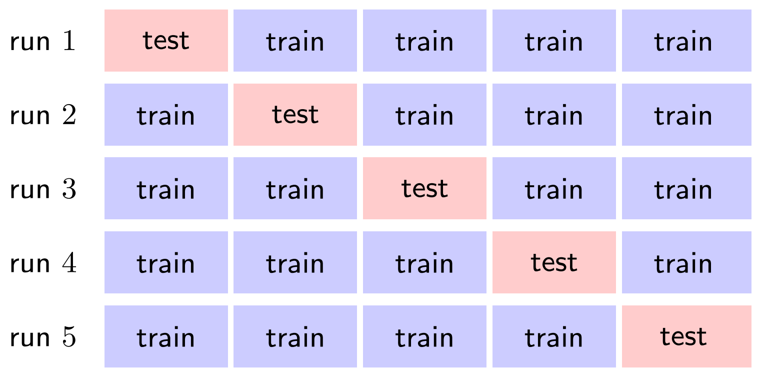

Figure 6.3: Illustration of k-fold cross-validation for $k = 5$. The data is partitioned into 5 partitions (folds). The computation of the prediction error is performed in 5 rounds. In each round, 4 partitions are used to train a set of models, which are then evaluated on the remaining partition. The performance scores from the 4 rounds are then averaged.

We will then choose the model (hypothesis class) that minimizies $L_{k-\mathsf{fold}}(f(\boldsymbol{x};\boldsymbol{\alpha}))$.

Typical values for k are 5 or 10. When data is scarce, we may consider the case $k = N$, which is referred to as leave-one-out cross-validation. Consider as an example 5-fold cross-validation, i.e., a 80/20 split of the data for training and validation, shown in Figure 6.3. In this case, we would train each model 5 times on 80% of the data and validate it on the remaining 20%. We ensure that each data point ends up in the 20% validation set (sometimes called the test set, not to be confused with the test set defined previously) exactly once. With this procedure, we use every data point we have to contribute to an understanding of how well our model performs the task of learning from some data and predicting some new data.

Again, we use ridge regression to clarify how k-fold cross-validation is performed.

Example 6.2 **(5-fold cross-validation for ridge regression)**

We consider 5-fold cross-validation for ridge regression. We proceed as follows.

1. The first step is to choose a set of regularization parameters λ, $\lambda \in \mathcal{L} = \{\lambda_1, \dots, \lambda_{\mathsf{r}}\}$, i.e., a set of hypothesis classes, and partition the data into 5 partitions $\kappa \in \mathcal{P} = \{1, \dots, 5\}$.

2. Then, we proceed to select the best hypothesis class—hyperparameter λ. We proceed as follows. For each $\lambda \in \mathcal{L}$ and partition $\kappa \in \mathcal{P}$, we learn the optimal $\boldsymbol{w}$ as

$$\boldsymbol{w}^*_{\lambda,\kappa} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{4N_\kappa} \left(y_i - w_0 - \sum_{i=1}^{d} x_{i,j} w_j \right)^2 + \lambda \sum_{j=1}^{d} w_j^2 \,,$$

where the learning is performed using the data points in all partitions except partition κ, with a total number of $4N_\kappa$ training points.

Based on partition κ, we then compute the generalization error

$$L_{\mathcal{V}}(\lambda, \kappa, \boldsymbol{w}^*_{\lambda,\kappa}) = \frac{1}{N_\kappa} \sum_{i=1}^{N_\kappa} \ell(y_i, f(\boldsymbol{x}_i; \lambda, \boldsymbol{w}^*_{\lambda,\kappa}))\,.$$

We then select λ that minimizes

$$L_{\mathcal{V}}(f(\boldsymbol{x};\lambda)) = \frac{1}{5}\sum_{\kappa=1}^{5} L_{\mathcal{V}}(\lambda,\kappa,\boldsymbol{w}^*_{\lambda,\kappa})\,,$$

i.e.,

$$\lambda^* = \arg\min_{\lambda\in\{\mathcal{L}\}} \; L_{\mathcal{V}}(f(\boldsymbol{x};\lambda))\,.$$

In words, we select λ that minimizes the estimate of the generalization error obtained by averaging the estimate of the generalization error over all 5 partitions.

3. We have now chosen the hypothesis class. Then, for selected hypothesis class, i.e., the selected λ, which we denote by λ^*, we retrain $\boldsymbol{w}$ based on all the data, i.e., all $5N_\kappa$ data points.

4. Finally, for chosen specific model, given by parameters $(\lambda^*, \boldsymbol{w}^*_{\lambda^*})$, we estimate the generalization error based on a test set $\mathcal{T}$ which we have set aside (this last step is sometimes avoided).

As k-fold cross-validation ensures that every observation from the original dataset has the chance of appearing in both the training and validation set, this method generally results in a less biased model compared to other methods. It is one of the best approaches if we have limited input data. A major drawback of k-fold cross-validation, however, is that the number of training runs is k, which may be a problem when training is computationally expensive.

Appendix A

Schmidt orthogonalization

Two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are orthogonal if their inner product is zero, i.e., $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\mathsf{T}\boldsymbol{b} = 0$ (we write $\boldsymbol{a} \perp \boldsymbol{b}$). We recall the Schmidt process for orthogonalization of vectors. We define the projection operator

$$\mathsf{proj}_{\boldsymbol{b}}(\boldsymbol{a}) = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\langle \boldsymbol{b}, \boldsymbol{b} \rangle}\boldsymbol{b} = \frac{\boldsymbol{a}^\mathsf{T}\boldsymbol{b}}{\|\boldsymbol{b}\|^2}\boldsymbol{b}\,,$$

which projects the vector $\boldsymbol{a}$ orthogonally onto the line spanned by vector $\boldsymbol{b}$. Then, we can obtain a vector $\tilde{\boldsymbol{a}}$ orthogonal to $\boldsymbol{b}$ as

$$\begin{aligned} \tilde{\boldsymbol{a}} &= \boldsymbol{a} - \mathsf{proj}_{\boldsymbol{b}}(\boldsymbol{a}) \\ &= \boldsymbol{a} - \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\langle \boldsymbol{b}, \boldsymbol{b} \rangle}\boldsymbol{b}\,. \end{aligned}$$

The length of the orthogonal projection of $\boldsymbol{a}$ onto $\boldsymbol{b}$ is given by $a = \|\boldsymbol{a}\| \cos\theta$, where θ is the angle between $\boldsymbol{a}$ and $\boldsymbol{b}$,

$$\cos\theta = \frac{\boldsymbol{a}\boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}\,.$$

Hence,

$$a = \frac{\boldsymbol{a}\boldsymbol{b}}{\|\boldsymbol{b}\|}\,.$$

Appendix B

Mean and variance

B.1 Sample mean, sample variance, and sample covariance matrix

Suppose that a population has N elements $x_1, \ldots, x_N$. Then, the sample average (or population mean) is given by

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i \,,$$

and the sample variance by

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 \,.$$

Consider N data points, each being a d-dimensional vector, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, where $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,p})^\mathsf{T}$, which we collect in the matrix

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,p} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,p} \\ \vdots & \vdots & \ldots & \vdots \\ x_{N,1} & x_{N,2} & \ldots & x_{N,p} \end{pmatrix} .$$

The sample covariance matrix of the data points is a $p \times p$ matrix

$$\begin{aligned} \boldsymbol{\Sigma} &= \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\mathsf{T} \\ &= \frac{1}{N} (\boldsymbol{X} - \bar{\boldsymbol{X}})^\mathsf{T} (\boldsymbol{X} - \bar{\boldsymbol{X}}) \,, \end{aligned} \tag{B.1}$$

where $\bar{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i$, i.e., the sample mean of the vectors.

The variance of a constant matrix $\boldsymbol{A}$ times a random vector x is

$$\mathrm{Var}[\boldsymbol{A}\mathsf{x}] = \boldsymbol{A} \mathrm{Var}(\mathsf{x}) \boldsymbol{A}^\mathsf{T}$$

B.2 Transpose properties

$$(\boldsymbol{A} + \boldsymbol{B})^\mathsf{T} = \boldsymbol{A}^\mathsf{T} + \boldsymbol{B}^\mathsf{T}$$

$$(\boldsymbol{AB})^\mathsf{T} = \boldsymbol{B}^\mathsf{T}\boldsymbol{A}^\mathsf{T}$$

$$(\boldsymbol{A}^{-1})^\mathsf{T} = (\boldsymbol{A}^\mathsf{T})^{-1}$$

$$\boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \boldsymbol{y}^\mathsf{T}\boldsymbol{x}$$

B.3 Orthogonal and semi-orthogonal matrices

An orthogonal matrix, or orthonormal matrix, is a real square matrix whose columns and rows are orthonormal vectors.[1] An orthogonal matrix satisfies

$$\boldsymbol{A}^\mathsf{T}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\mathsf{T} = \boldsymbol{I}$$

This leads to the equivalent characterization: a matrix $\boldsymbol{A}$ is orthogonal if its transpose is equal to its inverse,

$$\boldsymbol{A}^\mathsf{T} = \boldsymbol{A}^{-1}$$

A semi-orthogonal matrix is a non-square matrix with real entries where: if the number of columns exceeds the number of rows, then the rows are orthonormal vectors; but if the number of rows exceeds the number of columns, then the columns are orthonormal vectors.

If $\boldsymbol{A}$ is an $m \times n$ matrix for $m > n$, then

$$\boldsymbol{A}^\mathsf{T}\boldsymbol{A} = \boldsymbol{I}$$

B.4 Inverse of product of matrices

$$(\boldsymbol{ABC})^{-1} = \boldsymbol{C}^{-1}\boldsymbol{B}^{-1}\boldsymbol{A}^{-1} \tag{B.2}$$

B.5 Eigendecomposition of a matrix

Given am $n \times n$ square matrix $\boldsymbol{A}$, its eigendecompisition is

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}\,, \tag{B.3}$$

where $\boldsymbol{Q}$ is the $n \times n$ matrix whose columns are (right) eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix whose diagonal elements are the corresponding eigenvalues.

[1]Two vectors in an inner product space are orthonormal if they are orthogonal (or perpendicular along a line) unit vectors, i.e., with norm one, $\|\boldsymbol{a}\| = 1$.

B.6 ℓ_2 norm, ℓ_1 norm, ℓ_0 norm

The ℓ_2 norm of a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ is defined as

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}\,.$$

For notational simplicity, we will write $\|\boldsymbol{x}\|$ for the ℓ_2 norm, which is the one we will use the most. When the norm is not the ℓ_2 norm, we will indicate it explicitly.

The ℓ_1 norm of a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ is defined as

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|\,.$$

The ℓ_0 norm of a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ corresponds to the number of nonzero elements in the vector.

Bibliography

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data minining, inference, and prediction*, 2nd ed. Springer, 2008.

[2] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, 1st ed. Springer, 2013.