Machine Learning Handbook

Predrag Radivojac and Martha White

November 1, 2018

Table of Contents

Notation Reference

Set notation

$\mathcal{X}$ A generic set of values. For example, $\mathcal{X} = \{0, 1\}$ is the set containing only 0 and 1, $\mathcal{X} = [0, 1]$ is the interval from 0 to 1 and $\mathcal{X} = \mathbb{R}$ is the set of real numbers. Depending on occasion, symbols such as A, B, Ω, and others will also be used as sets.

$\mathcal{P}(\mathcal{X})$ The power set of $\mathcal{X}$, a set containing all possible subsets of $\mathcal{X}$.

$[a, b]$ Closed interval with $a < b$, including both a and b.

(a, b) Open interval with $a < b$, with neither a nor b in the set.

$(a, b]$ Open-closed interval with $a < b$, including b but not a.

$[a, b)$ Closed-open interval with $a < b$, including a but not b.

Vector and matrix notation

x Unbold lowercase variables are generally scalars. However, when $x \in \mathcal{X}$, where $\mathcal{X}$ is not specified, x may imdicate a vector, a structured object such as graph, etc.

$\mathbf{x}$ Bold lowercase variables are vectors. By default, vectors are column vectors.

$\mathbf{X}$ Bold uppercase variables are matrices. This looks like a multivariate random variable, $\boldsymbol{X}$, but the random variable is italicized. It will often be clear from context when this is a multivariate random variable and when it is a matrix.

$\mathbf{X}^\top$ The transpose of the matrix. For two matrices $\mathbf{A}$ and $\mathbf{B}$, it holds that

$$(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top.$$

An $n \times d$ matrix consisting of n vectors each of dimension d can be expressed as

$$\mathbf{X} = [\mathbf{x}_1 \, \mathbf{x}_2 \, \ldots \, \mathbf{x}_n]^\top.$$

$\mathbf{X}_{i:}$ The i-th row of the matrix. A row vector.

$\mathbf{X}_{:j}$ The j-th column of the matrix. A column vector.

Tuples, vectors, and sequences

$(x_1, x_2, \ldots, x_d)$ A tuple; i.e., an ordered list of d elements. When $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, the tuple will be treated as a column vector $\mathbf{x} = [x_1\ x_2\ \ldots\ x_d]^\top$.

$a_1, \ldots, a_m$ A sequence of m items. Index variables over these sequences are usually the variables i, j, or k. For example, $\sum_{i=1}^{m} a_i$ or, if each $\mathbf{a}_i$ is a vector of dimension d, then the double index $\sum_{i=1}^{m} \sum_{j=1}^{d} a_{ij}$.

Function notation

$f : \mathcal{X} \to \mathcal{Y}$ The function is defined on domain $\mathcal{X}$ to co-domain $\mathcal{Y}$, taking values $x \in \mathcal{X}$ and sending them to $f(x) \in \mathcal{Y}$.

$\frac{df}{dx}(x)$ The derivative of a function at $x \in \mathcal{X}$, where $f : \mathcal{X} \to \mathbb{R}$ for $\mathcal{X} \subset \mathbb{R}$.

$\nabla f(\mathbf{x})$ The gradient of a function at $\mathbf{x} \in \mathcal{X}$, where $f : \mathcal{X} \to \mathbb{R}$ for $\mathcal{X} \subset \mathbb{R}^d$. It holds that

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_d} \right).$$

$H_{f(\mathbf{x})}$ The Hessian matrix of a function at $\mathbf{x} \in \mathcal{X}$, where $f : \mathcal{X} \to \mathbb{R}$ for $\mathcal{X} \subset \mathbb{R}^d$. It holds that

$$H_{f(\mathbf{x})} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & & \\ \vdots & & \ddots & \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & & & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}.$$

Random variables and probabilities

X A univariate random variable is written in uppercase.

$\mathcal{X}$ The space of values for the random variable.

x Lowercase variable is an instance or outcome, $x \in \mathcal{X}$.

$\boldsymbol{X}$ A multivariate random variable is written bold uppercase.

$\boldsymbol{x}$ Lowercase bold variable is a multivariate instance. In particular cases, when the variable value is treated as a vector, we will use $\mathbf{x}$.

$\mathcal{N}(\mu, \sigma^2)$ A univariate Gaussian distribution, with parameters μ, σ^2.

$\sim$ indicates that a variable is distributed as e.g., $X \sim \mathcal{N}(\mu, \sigma^2)$.

Parameters and estimation

$\mathcal{D}$ A data set, typically composed of n elements of multivariate inputs $\mathbf{X} \in \mathbb{R}^{n \times d}$ and univariate outputs $\mathbf{y} \in \mathbb{R}^n$ or multivariate outputs $\mathbf{Y} \in \mathbb{R}^{n \times m}$. The data set will also be referred to as a set of indexed tuples; i.e., $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$.

M Represents a generic model, to discuss general parameter estimation. For example, $M = \theta$ for some parameters θ, such as the mean of Gaussian distribution.

$\boldsymbol{\omega}$ The true parameters for the (generalized) linear regression and classification models, typically with $\boldsymbol{\omega} \in \mathbb{R}^d$.

$\mathbf{w}$ The approximated parameters for the (generalized) linear regression and classification models, typically with $\mathbf{w} \in \mathbb{R}^d$. When discussing $\mathbf{w}$ as the maximum likelihood solution on some data, we write $\mathbf{w}_{\text{ML}}(\mathcal{D})$, to indicate that the variability arises from $\mathcal{D}$.

$\max_{a \in \mathcal{B}} f(a)$ The maximum value of a function f across values a in a set $\mathcal{B}$.

$\text{argmax}_{a \in \mathcal{B}} f(a)$ The item a in set $\mathcal{B}$ that produces the maximum value $f(a)$.

Norms

$\|\mathbf{x}\|$ A norm on $\mathbf{x}$.

$\|\mathbf{x}\|_2$ The ℓ_2 norm on a vector, $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^d x_i^2}$. This norm gives the Euclidean distance from the origin of the coordinate system to $\mathbf{x}$; that is, it is the length of vector $\mathbf{x}$.

$\|\mathbf{x}\|_2^2$ The squared ℓ_2 norm on a vector, $\|\mathbf{x}\|_2^2 = \sum_{i=1}^d x_i^2$.

$\|\mathbf{x}\|_p$ The general ℓ_p norm on a vector, $\|\mathbf{x}\|_p = (\sum_{i=1}^d |x|_i^p)^{1/p}$.

$\|\mathbf{X}\|_F$ The Frobenius norm of an n-by-d matrix; i.e.,

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^d X_{ij}^2} = \sqrt{\sum_{i=1}^n \|X_{i:}\|_2^2} = \sqrt{\sum_{j=1}^d \|X_{:j}\|_2^2}.$$

Useful formulas and rules

$$\log\left(\frac{x}{y}\right) = \log(x) - \log(y)$$

$$\log\left(x^y\right) = y\log(x)$$

$$\sum_{i=1}^{m} a_i \int_{\mathcal{X}} f_i(x)p(x)dx = \int_{\mathcal{X}} \sum_{i=1}^{m} a_i f_i(x)p(x)dx \quad \triangleright \text{ Can bring sum into integral}$$

$$\frac{d}{dx}\int_{\mathcal{X}} f(x)p(x)dx = \int_{\mathcal{X}} \frac{d}{dx} f(x)p(x)dx \quad \triangleright \text{ Can (almost always) bring derivative into integral}$$

Preface: A starting example with linear regression

Machine learning involves a broad range of techniques for learning from data; a central goal — and the one we largely discuss in this handbook — is prediction. Many learning techniques are centered around learning a function $f : \mathbb{R}^d \to \mathbb{R}$ that inputs attributes or features about an item, and produces an output prediction about that item. For example, consider a setting where you would like to guess or predict the price of a house based on information about that house. You might have features such as its age, the size of the house and, of course, distance to the nearest bakery. Without any previous examples of house costs, i.e., without any data, it might be hard to guess this price. However, imagine you are given a set of house features and the corresponding selling costs, for houses that sold this year. Let $\mathbf{x} \in \mathbb{R}^d$ be a vector of the features for a house, in this case $\mathbf{x} = [x_1\ x_2\ x_3]$ =[age, size, distance_to_bakery] and the target y = price. If we have 10 examples or instances of previous house prices, we have a dataset: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{10}, y_{10})$, where $(\mathbf{x}_i, y_i)$ is the feature-price pair for the ith house in your set of instances. A natural goal is to find a function f that accurately recreates the data, for example by trying to find a function f that results in a small difference between the prediction, $f(\mathbf{x}_i)$, and the actual price, y_i, for each house.

We can formalize this as an optimization problem. Imagine we have some space of possible functions, $\mathcal{H}$, from which we can select our function f. For a simple case, let us imagine that the function is linear: $f(\mathbf{x}) = w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$ for any $\mathbf{w} = [w_0\ w_1\ w_2\ w_3] \in \mathbb{R}^d$ where w_0 is the intercept of the linear function. We can try to find a function from the class of linear functions that minimizes these squared differences

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{10} (f(\mathbf{x}_i) - y_i)^2$$

As we will see later in Chapter 5, this optimization problem is simple to solve for linear functions. The procedure involves explicitly writing the optimization in terms of the parameters $\mathbf{w}$ and solving for the optimal $\mathbf{w}$ that makes the differences as small as possible

$$\begin{aligned} \mathrm{Err}(\mathbf{w}) &= \sum_{i=1}^{10} \left(f(\mathbf{x}_i) - y_i\right)^2 \\ &= \sum_{i=1}^{10} \left(w_0 + \sum_{j=1}^{d} w_j x_{ij} - y_i \right)^2 . \end{aligned}$$

The solution is a straight line that tries to best fit the observed targets y. A simple illustration of such a function, for only one attribute, is depicted in Figure 1.

Once we have this function, when we see a new house, we hope that it is similar enough to the previous houses so that this function adequately predicts its house price. The learned function f interpolates between these 10 points to predict on unseen points. But, a natural question is, did we interpolate well and is the learned f going to produce an accurate

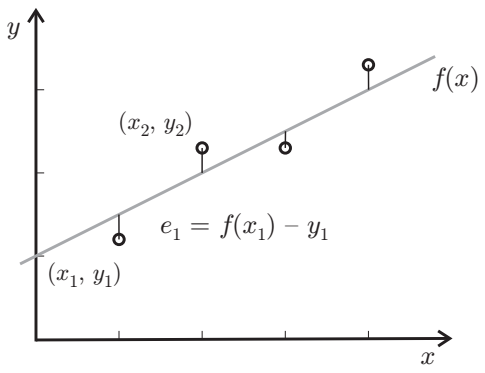


Figure 1: An example of a linear regression fitting on data set $\mathcal{D} = \{(1, 1.2), (2, 2.3), (3, 2.3), (4, 3.3)\}$. *The task of the optimization process is to find the best linear function* $f(x) = w_0 + w_1 x$ *so that the sum of squared errors* $e_1^2 + e_2^2 + e_3^2 + e_4^2$ *is minimized.*

prediction on new houses? If you want to use this learned function f in practice, you want to have such a characterization.

In the agnostic development above, it is difficult to answer such questions. We can make intuitive modifications that we hope will provide more accurate predictions, like extending the class of functions to complex non-linear functions. But, these functional modifications still do not help characterize accuracy of the prediction on new houses. Rather, what we are missing is a notion of confidence. How confident are we in the predictions? Did we see enough previous houses to be confident about this prediction? What is the source of variability? How do we deal with variability? All these types of questions require a probabilistic treatment.

In this handbook, we start by providing an introduction to probability, to provide a base for dealing with uncertainty in machine learning. We then return to learning these functions, once we have the probabilistic tools to better understand how to approach the answers to these questions. Much of the required mathematical background will involve basic understanding of probability and optimization; this handbook will attempt to provide most of that required background throughout.

Chapter 1

Introduction to Probabilistic Modeling

Modeling the world around us and making predictions about the occurrence of events is a multidisciplinary endeavor standing on the solid foundations of probability theory, statistics, and computer science. Although intertwined in the process of modeling, these fields have relatively discernible roles and can be, to a degree, studied individually. Probability theory brings the mathematical infrastructure, firmly grounded in its axioms, for manipulating probabilities and equips us with a broad range of models with well-understood theoretical properties. Statistics contributes frameworks to formulate inference and the process of narrowing down the model space based on the observed data and our experience in order to find, and then analyze, solutions. Computer science provides us with theories, algorithms, and software to manage the data, compute the solutions, and study the relationship between solutions and available resources (time, space, computer architecture, etc.). As such, these three disciplines form the core quantitative framework for all of empirical science and beyond.

Probability theory and statistics have a relatively long history; the formal origins of both can be traced to the 17^{th} century. Probability theory was developed out of efforts to understand games of chance and gambling. The correspondence between Blaise Pascal and Pierre de Fermat in 1654 serves as the oldest record of modern probability theory. Statistics, on the other hand, originated from data collection initiatives and attempts to understand trends in the society (e.g., manufacturing, mortality causes, value of land) and political affairs (e.g., public revenues, taxation, armies). The two disciplines started to merge in the 18^{th} century with the use of data for inferential purposes in astronomy, geography, and social sciences. The increased complexity of models and availability of data in the 19^{th} century emphasized the importance of computing machines. This contributed to establishing the foundations of the field of computer science in the 20^{th} century, which is generally attributed to the introduction of the von Neumann architecture and formalization of the concept of an algorithm. The convergence of the three disciplines has now reached the status of a principled theory of probabilistic inference with widespread applications in science, business, medicine, military, political campaigns, etc. Interestingly, various other disciplines have also contributed to the core of probabilistic modeling. Concepts such as a Boltzmann distribution, a genetic algorithm, or a neural network illustrate the influence of physics, biology, psychology, and engineering.

We will refer to the process of modeling, inference, and decision making based on probabilistic models as *probabilistic reasoning* or reasoning under uncertainty. Some form of reasoning under uncertainty is a necessary component of everyday life. When driving, for example, we often make decisions based on our expectations about which way would be best to take. While these situations do not usually involve an explicit use of probabilities and probabilistic models, an intelligent driverless car such as Google Chauffeur must make

use of them. And so must a spam detection software in an email client, a credit card fraud detection system, or an algorithm that infers whether a particular genetic mutation will result in disease. Therefore, we first need to understand the concept of probability and then introduce a formal theory to incorporate evidence (e.g., data collected from instruments) in order to make good decisions in a range of situations. At a basic level, probabilities are used to quantify the chance of the occurrence of events. As Jacob Bernoulli brilliantly put it in his work *The Art of Conjecturing* (1713), "[p]robability, [...] is the degree of certainty, and it differs from the latter as a part differs from the whole". He later adds, "To make a conjecture [prediction] about something is the same as to measure its probability. Therefore, we define the art of conjecturing [science of prediction] or stochastics, as the art of measuring probabilities of things as accurately as possible, to the end that, in judgements and actions, we may always choose or follow that which has been found to be better, more satisfactory, safer, or more carefully considered." The techniques of probabilistic modeling formalize many intuitive concepts. In a nutshell, they provide toolkits for rigorous mathematical analysis and inference, often in the presence of evidence, about events influenced by factors that we either do not fully understand or have no control of.

To provide a quick insight into the concept of uncertainty and modeling, consider rolling a fair six-sided die. We could accurately predict, or so we think, the outcome of a roll if we carefully incorporated the initial position, force, friction, shape defects, and other physical factors and then executed the experiment. But the physical laws may not be known, they can be difficult to incorporate or such actions may not even be allowed by the rules of the experiment. Thus, it is practically useful to simply assume that each outcome is equally likely; in fact, if we rolled the die many times, we would indeed observe that each number is observed roughly equally. Assigning an equal chance (probability) to each outcome of the roll of a die provides an efficient and elegant way of modeling uncertainties inherent to the experiment.

Another, more realistic example in which collecting data provides a basis for simple probabilistic modeling is a situation of driving to work every day and predicting how long it will take us to reach the destination tomorrow. If we recorded the "time to work" for a few months we would observe that trips generally took different times depending on many internal (e.g., preferred speed for the day) and also external factors (e.g., weather, road works, encountering a slow driver). While these events, if known, could be used to predict the exact duration of the commute, it is unrealistic to expect to have full information—rather we have *partial observability*. It is useful to provide ways of aggregating external factors via collecting data over a period of time and providing the distribution of the commute time. Such a distribution, in the absence of any other information, would then facilitate reasoning about events such as making it on time to an important meeting at 9 am.

The techniques of probabilistic modeling provide a formalism for dealing with such repetitive experiments influenced by a number of external factors over which we have little control or knowledge. With such a formalism, we can better understand and improve how we make predictions, because we can more clearly specify our assumptions about our uncertainty and explicitly reason about possible outcomes. In this chapter, we introduce probabilities and probability theory, from the beginning. Because probability is such a fundamental concept in machine learning, it is worth understanding where it comes from. Nonetheless, following the spirit of these notes, the treatment will be brief and focus mostly on what is needed to understand the development in following chapters.

1.1 Probability Theory and Random Variables

Probability theory is as a branch of mathematics that deals with measuring the likelihood of events. At the heart of probability theory is the concept of an *experiment*. An experiment can be the process of tossing a coin, rolling a die, checking the temperature tomorrow or figuring out the location of one's keys. When carried out, each experiment has an *outcome*, which is an element drawn from a set of predefined options, potentially infinite in size. The outcome of a roll of a die is a number between one and six; the temperature tomorrow might be a real number; the outcome of the location of one's keys can be a discrete set of places such as a kitchen table, under a couch, in office etc. In many ways, the main goal of probabilistic modeling is to formulate a particular question or a hypothesis pertaining to the physical world as an experiment, collect the data, and then construct a model. Once a model is created, we can compute quantitative measures of sets of outcomes we are interested in and assess the confidence we should have in these measures.

We can build up rules of probability, based on an elegantly simple set of axioms called the *axioms of probability*. Let the *sample space* (Ω) be a non-empty set of outcomes and the *event space* ($\mathcal{E}$) be a non-empty set of subsets of Ω. For example, $\Omega = \{1, 2, 3\}$ and one possible *event* is $A = \{1, 3\} \in \mathcal{E}$, where the event is that a 1 or a 3 is observed. The event space $\mathcal{E}$ is a *set of measurable events* if[1]

1. $A \in \mathcal{E} \quad \Rightarrow \quad A^c \in \mathcal{E}$
2. $A_1, A_2, \ldots \in \mathcal{E} \quad \Rightarrow \quad \bigcup_{i=1}^{\infty} A_i \in \mathcal{E}$

where A and all A_i's are events and A^c is the complement of A; i.e., $A^c = \Omega - A$. If $\mathcal{E}$ satisfies these two properties, $(\Omega, \mathcal{E})$ is said to be a *measurable space*. Now we can define the axioms of probability, which make it more clear why these two conditions are needed for our event space to define meaningful probabilities over events. A function $P : \mathcal{E} \to [0, 1]$ satisfies the axioms of probability if

1. $P(\Omega) = 1$
2. $A_1, A_2, \ldots \in \mathcal{E},\ A_i \cap A_j = \varnothing\ \forall i, j \quad \Rightarrow \quad P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$

is called a *probability measure* or a *probability distribution*. The tuple $(\Omega, \mathcal{A}, P)$ is called the *probability space*.

The beauty of these axioms lies in their compactness and elegance. Many useful expressions can be derived from the axioms of probability. For example, it is obvious that $P(A^c) = 1 - P(A)$. This makes it more clear why we required that if an event is in the event space, then its complement should also be in the event space: if we can measure the probability of an event, then we know that the probability of that event not occurring is 1 minus that probability. Similarly, we require that if two events A_1, A_2 are disjoint, then $P(A_1 \cup A_2) = P(A_1) + P(A_2)$: the probability of either event occurring is the sum of their probabilities, because there is no overlap in the outcomes in the events. Another property we can infer is that we always have $\Omega, \varnothing \in \mathcal{E}$, where $\varnothing$ corresponds to the event where

[1] Such a set is usually called a *sigma algebra* or *sigma field*. This terminology feel daunting and is only due to historical naming conventions in German. Because the sigma algebra is simply the set of events to which we can assign probabilities (measure), we will use the longer but more clear name.

nothing occurs—which must have zero probability. If you are interested in other rules that can be derived, see Appendix A.1.

Example 1: [Discrete variables (countable)] Consider modeling the probabilities of the roll of a dice. The outcome space is the finite set $\Omega = \{1, 2, 3, 4, 5, 6\}$ and the event space $\mathcal{E}$ is the power set $\mathcal{P}(\Omega) = \{\varnothing, \{1\}, \{2\}, \ldots, \{2, 3, 4, 5, 6\}, \Omega\}$, which consists of all possible subsets of Ω. A natural probability distribution on $(\Omega, \mathcal{E})$ gives each dice roll a $1/6$ chance of occurring, defined as $P(\{x\}) = 1/6$ for $x \in \Omega$, $P(\{1, 2\}) = 1/3$ and so on. □

Example 2: [Continuous variables (uncountable)] Consider modeling the probabilities of the stopping time of a car, in the range of 3 seconds to 6 seconds. The outcome space is the continuous interval $\Omega = [3, 6]$. An event could be that the car stops within 3 to 3.1 seconds, giving $A = [3, 3.1] \in \mathcal{E}$. The probability $P(A)$ of such an event is likely low, because it would be a very fast stopping time. We could then start considering all possible time intervals for the event space, and corresponding probabilities. We can already see that this will be a bit more complicated for continuous variables, and so we more rigorously show how to define $\mathcal{E}$ and P below in Section 1.2.2. □

These two examples demonstrate the two most common cases we will encounter: discrete variables and continuous variables. The terms above—countable and uncountable—indicate whether a set can be enumerated or not. For example, the set of natural numbers can be enumerated, and so is countable, whereas the set of real numbers cannot be enumerated—there is always another real number between any two real numbers—and so is uncountable. Though this distinction results in real differences—such as using sums for countable sets and integrals for uncountable sets—the formalism and intuition will largely transfer between the two settings. We will focus mostly on discrete and continuous variables. Much of the same ideas also transfer to mixed variables, where outcome spaces are composed of both discrete and continuous sets such as $\Omega = [0, 1] \cup \{2\}$. Further, for the uncountable setting, we specifically discuss continuous sets, i.e., those are unions of continuous intervals such as $\Omega = [0, 1] \cup [5, 10]$. Because almost all uncountable sets that we will want to consider are continuous, we will interchangeably use the terms continuous and uncountable to designate such spaces. Finally, discrete sets can either be finite, such as $\{1, 2, 3\}$, or countably infinite, such as the natural numbers. Continuous sets are clearly infinite, and are said to be uncountably infinite.

Before going further in-depth on how to define probability distributions, we first introduce *random variables*, and from here on will deal strictly with random variables. A random variable lets us more rigorously define transformations of probability spaces; once we execute that transformation, we can forget about the underlying probability space and can focus on the events and distribution only on the random variable. This is in fact what you do naturally when defining probabilities over variables, without needing to formalize it mathematically. Of course, here we will formalize it.

Consider again the dice example, where now instead you might want to know: what is the probability of seeing a low number (1-3) or a high number (4-6)? We can define a new probability space with $\Omega_X = \{\text{low}, \text{high}\}$, $\mathcal{E}_X = \{\varnothing, \{\text{low}\}, \{\text{high}\}, \Omega\}$ and $P_X(\{\text{low}\}) =$

$1/2 = P_X(\{\text{high}\})$. The transformation function $X : \Omega \to \Omega_X$ is defined as

$$X(\omega) = \begin{cases} \text{low} & \text{if } \omega \in \{1, 2, 3\} \\ \text{high} & \text{if } \omega \in \{4, 5, 6\} \end{cases}$$

The distribution P_X is immediately determined from this transformation. For example, $P_X(\{\text{low}\}) = P(\{\omega : X(\omega) = \text{low}\})$, because the underlying probability space indicates the likelihood of seeing a $1, 2$ or 3. Now we can answer questions about the probability of seeing a low number of a higher number.

This function X is called a random variable. It is slightly confusing that it is neither random, nor a variable, since X is a function. However, from this point onward, we will treat X like it is a variable that is random, by more simply writing statements like $P_X(X = x)$ or $P_X(X \in A)$, rather than $P(\{\omega : X(\omega) = x\})$ or $P(\{\omega : X(\omega) \in A\})$. For correctness, we can remember that it is a function defined on a more complex underlying probability space. But, in practice, we can start thinking directly in terms of the random variable X and the associated probabilities. Similarly, even for the dice role, we can acknowledge that there is a more complex underlying probability space, defined by the dynamics of the dice. When considering only the probabilities of discrete outcomes from 1-6, we have already implicitly applied a transformation on top of probabilities of the physical system.

Once we have a random variable, it defines a valid probability space $(\Omega_X, \mathcal{E}_X, P_X)$. Therefore, all the same rules of probability apply, the same understanding of how to define distributions, etc. In fact, we can always define a random variable X that corresponds to no transformation, to obtain the original probability space. For this reason, we can move forward assuming we are always dealing with random variables, without losing any generality. We will drop the subscripts, and consider $(\Omega, \mathcal{E}, P)$ to be defined for X. For a more in-depth discussion on random variables, see Appendix A.4.

1.2 Defining distributions

Now we would like to know how to specify P to satisfy the axioms of probability, to model the probability of X taking values in an event A, $P(X \in A)$ with outcome space Ω. This task feels daunting, because it seems we need to define the likelihood for every possible event—set of outcomes—and, in such a way that satisfies the axioms of probability, no less! Fortunately, instead we can define the distribution using a function defined directly on instances $x \in \Omega$. It is convenient to separately consider discrete (countable) and continuous (uncountable) sample spaces. For the discrete case, we will define probability mass functions and for the continuous case, we will define probability density functions.

1.2.1 Probability mass functions

Let Ω be a discrete sample space and $\mathcal{E} = \mathcal{P}(\Omega)$, the power set of Ω. A function $p : \Omega \to [0, 1]$ is called a *probability mass function* (pmf) if

$$\sum_{\omega \in \Omega} p(\omega) = 1.$$

The probability of any event $A \in \mathcal{E}$ is defined as

$$P(A) = \sum_{\omega \in A} p(\omega).$$

It is straightforward to verify that P satisfies the axioms of probability and, thus, is a probability distribution. For discrete random variables, therefore, we will often write $P(X = x)$, which means that $P(X = x) = p(x)$ for each outcome $x \in \Omega$. We rarely, if ever, define the distribution directly, and rather define the pmf p which induces the distribution P.

Example 3: Consider a roll of a fair six-sided die; i.e., $\Omega = \{1, 2, 3, 4, 5, 6\}$, and the event space $\mathcal{E} = \mathcal{P}(\Omega)$. What is the probability that the outcome is a number greater than 4?

First, because the die is fair, we know that $p(\omega) = \frac{1}{6}$ for $\forall \omega \in \Omega$. Now, let A be an event in $\mathcal{E}$ that the outcome is greater than 4; i.e., $A = \{5, 6\}$. Thus,

$$P(A) = \sum_{\omega \in A} p(\omega) = \frac{1}{3}.$$

Notice that the distribution P is defined on the elements of $\mathcal{E}$, whereas p is defined on the elements of Ω. That is, $P(\{1\}) = p(1)$, $P(\{2\}) = p(2)$, $P(\{1, 2\}) = p(1) + p(2)$, etc. □

To specify P, therefore, we need to determine how to specify the pmf, i.e., the probability of each discrete outcome. The pmf is often specified as a table of probability values. For example, to model the probability of a birthday for each day in the year, one could have a table of 365 values between zero and one, as long as the probabilities sum to 1. These probabilities could be computed from data about individuals birthdays, using counts for each day and normalizing by the total number of people in the population to estimate the probability of seeing a birthday on a given day. Such a table of values is very flexible, allowing precise probability values to be specified for each outcome. There are, however, a few useful pmfs that have a (more restricted) functional form. We describe three such pmfs here, that we will use throughout this handbook; for more examples of pmfs, see Appendix A.2.

The *Bernoulli distribution* derives from the concept of a Bernoulli trial, an experiment that has two possible outcomes: success and failure. In a Bernoulli trial, a success occurs with probability α and, thus, failure occurs with probability $1 - \alpha$. A toss of a coin (heads/tails), a basketball game (win/loss), or a roll of a die (even/odd) can all be seen as Bernoulli trials. We model this distribution by setting a sample space to two elements and defining the probability of one of them as α. More specifically, $\Omega = \{\text{success}, \text{failure}\}$ and

$$p(\omega) = \begin{cases} \alpha & \omega = \text{success} \\ 1 - \alpha & \omega = \text{failure} \end{cases}$$

where $\alpha \in (0, 1)$ is a parameter. If we take instead that $\Omega = \{0, 1\}$, we can compactly write the Bernoulli distribution as $p(\omega) = \alpha^{\omega}(1 - \alpha)^{1-\omega}$ for $\omega \in \Omega$. The Bernoulli distribution is often written Bernoulli(α). As we will see, a common setting where we use the Bernoulli is for binary classification, say where we try to predict whether a patient has the flu (outcome 0) or does not have the flue (outcome 1).

The *uniform distribution* for discrete sample spaces is defined over a finite set of outcomes each of which is equally likely to occur. Let $\Omega = \{1, \ldots, n\}$; then for $\forall \omega \in \Omega$

$$p(\omega) = \frac{1}{n}.$$

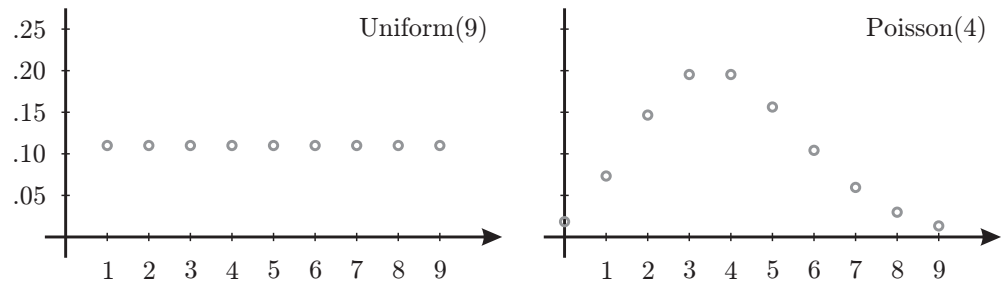


Figure 1.1: Two probability mass functions, for discrete random variables. The Poisson distribution continues further on the x-axis (for variable $\omega \in \mathbb{N}$), with probability decreasing to zero as $\omega \to \infty$.

The uniform distribution does not contain parameters; it is defined by the size of the sample space. We refer to this distribution as Uniform(n). We will see later that the uniform distribution can also be defined over finite intervals in continuous spaces.

The *Poisson distribution* reflects the probability of how many incidences occur (implicitly within a fixed time interval). For example, a call center is likely to receive 50 calls per hour, with a much smaller probability on only receiving 5 calls or receiving as many as 1000 calls. This can be modeled with a Poisson(λ), where λ represents the expected number of calls. More formally, $\Omega = \{0, 1, \ldots\}$ and for $\forall \omega \in \Omega$

$$p(\omega) = \frac{\lambda^\omega e^{-\lambda}}{\omega!}.$$

This mass function is hill-shaped, where the top of the hill is mostly centered around λ and there is a skew to having a short, steep left side of the hill and a long, less-steep right tail to the hill. The Poisson distribution is defined over an infinite sample space, but still countable. This is depicted in Figure 1.1.

Exercise 1: Prove that $\sum_{\omega \in \mathbb{N}} p(\omega) = 1$ for the Poisson distribution. □

Example 4: As a prelude to estimating parameters to distributions, consider an example of how we might use a Bernoulli distribution and determine the parameter α to the Bernoulli. A canonical example for Bernoulli distributions is a coin flip, where the outcomes are heads (H) or tails (T). $P(X = \text{H}) = \alpha$ is the probability of seeing H and $P(X = \text{T}) = 1 - \alpha$ is the probability of seeing T. We commonly assume $\alpha = 0.5$; this is called a fair (unbiased) coin. If we flipped the coin many times, we would expect to see about the same number of H and T. However, a *biased* coin may have some skew towards H or T. If we flipped the coin many times, if $\alpha > 0.5$ we should eventually notice more H come up and if $\alpha < 0.5$, we should notice more T.

How might we actually determine this α? An intuitive idea is to use repeated experiments (data), just as described above: flip the coin many times to see if you can gauge the skew. If you see 1000 H and 50 T, a natural guess for the bias is $\alpha = \frac{1000}{1000+50} \approx 0.95$. How confident are you in this solution? Is it definitely 0.95? And how do we more formally define why this should be the solution? This is in fact a reasonable solution, and corresponds to the maximum likelihood solution, as we will discuss in Chapter 3. □

1.2.2 Probability density functions

The treatment of continuous probability spaces is analogous to that of discrete spaces, with probability density functions (pdfs) replacing probability mass functions and integrals replacing sums. In defining pdfs, however, we will not be able to use tables of values, and will be restricted to functional forms. The main reason for this difference stems from the fact that it no longer makes sense to measure the probability of a singleton event. Consider again the stopping time for a car, discussed in Example 2. It would not make a lot of sense to ask the probability of the car stopping in exactly 3.14159625 seconds; realistically, the probability of such a precise event is vanishingly small. In fact, the probability of seeing precisely that stopping time is zero, because the set $\{3.14159625\}$ as a subset of $[3, 6]$ is a set of measure zero. Essentially, it takes up zero mass inside the interval $[3, 6]$, which is after all uncountably infinite. Instead, we will have to consider the probabilities of intervals, like $[4, 5]$ or $[5.667, 5.668]$.

For continuous spaces, we will assume that the set of events $\mathcal{E}$ consists of all possible intervals, called the Borel field $\mathcal{B}(\Omega)$. For example, if $\Omega = \mathbb{R}$, the Borel field $\mathcal{B}(\mathbb{R})$ consists of all open intervals (e.g., $(0, 1)$), closed intervals (e.g.,$[0, 1]$) and semi-open intervals (e.g., $[0, 1)$) in $\mathbb{R}$, as well as sets that can be obtained by a countable number of basic set operations on them, such as unions. This results in a more restricted set of events than the power set of Ω, which would, for example, include sets with only a singleton event. $\mathcal{B}(\mathbb{R})$ is still a huge set—an uncountably infinite set—but still smaller than $\mathcal{P}(\mathbb{R})$. Nicely, though, $\mathcal{B}(\mathbb{R})$ still contains all sets we could conceivably want to measure. The Borel field can be defined for any measurable space, such as higher-dimensional spaces like $\Omega = \mathbb{R}^2$, with events such as $A = [0, 1] \times [0, 1] \subset \Omega$ or $A = [1, 2] \times [-1, 4] \cup [0, 0.1] \times [10, 1000]$.

Let now Ω be a continuous sample space and $\mathcal{E} = \mathcal{B}(\Omega)$. A function $p : \Omega \to [0, \infty)$ is called a *probability density function* (pdf) if[2]

$$\int_{\Omega} p(\omega) d\omega = 1.$$

The probability of an event $A \in \mathcal{B}(\Omega)$ is defined as

$$P(A) = \int_{A} p(\omega) d\omega.$$

Notice that the definition of the pdf is not restricted to having a range $[0, 1]$, but rather to $[0, \infty)$. For pmfs, the probability of a singleton event $\{\omega\}$ is the value of the pmf at the sample point ω; i.e., $P(\{\omega\}) = p(\omega)$. Since probability distributions P are restricted to the range $[0, 1]$, this implies pmfs must also be restricted to that range. In contrast, the value of a pdf at point ω is not a probability; it can actually be greater than 1. Though it is common when speaking informally to call $p(x)$ the probability of x, more accurately we call this the density at x because it is most definitely not a probability. In fact, as mentioned above, the probability at any single point is 0 (i.e., a countable subset of Ω is a set of measure zero).

A natural confusion is how p can integrate to 1, but actually have values larger than 1. The reason for this is that $p(x)$ can be (much) larger than 1, as long as its only for a very

[2]For correctness, we would like to note that this definition uses Lebesgue integration. You do not need to know about nuanced differences in integration formulations; for all settings we consider, all the definitions are equivalent and your knowledge of integration rules will be effective.

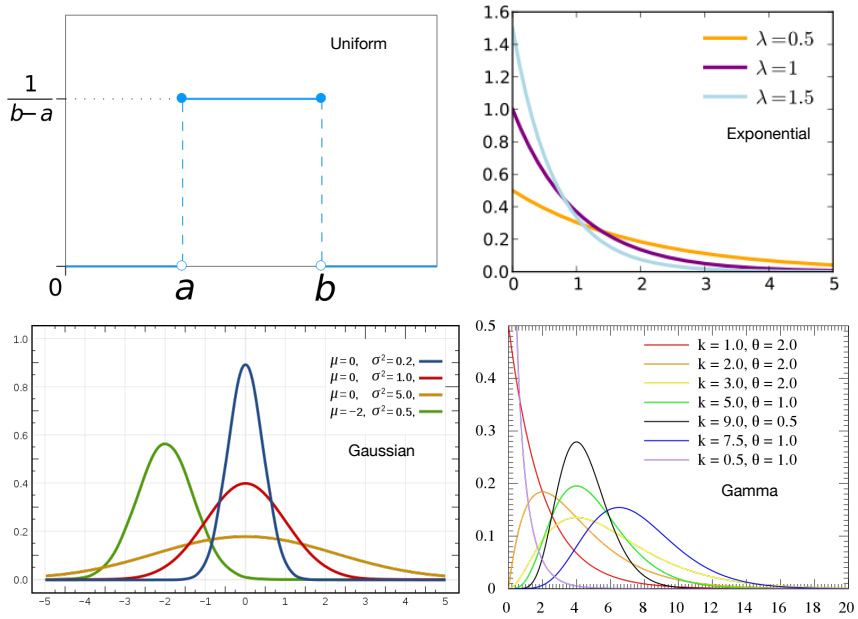


Figure 1.2: Four probability density functions, for continuous random variables. Images taken from Wikipedia.[3]

small interval. Consider the small interval $A = [x, x + \Delta x]$, with probability

$$\begin{aligned} P(A) &= \int_x^{x+\Delta x} p(\omega) d\omega \\ &\approx p(x)\Delta x. \end{aligned}$$

A potentially large value of the density function is compensated for by the small interval Δx to result in a number between 0 and 1. So, even if $p(x)$ is a million, and the density of points in the small interval or ball around x is large, the probability of an event must still be ≤ 1. The density does indicate that there is high likelihood around that point. By having a huge density around x, this suggests that the density for other points is zero or near zero and that the pdf is extremely peaked around x.

Unlike pmfs, we cannot so easily define pdfs p to flexibly provide specific probabilities for each outcome with a table of probabilities. Rather, for pdfs, we will usually use a known pdf that satisfies the required properties. Further, unlike the discrete case, we will never write $P(X = x)$, because that would be zero. Rather, we will typically write $P(X \in A)$ or more explicitly probabilistic questions like $P(X \leq 5)$. We highlight four pdfs here, that will be used throughout this book; for more examples of pdfs, see Appendix A.3.

The *uniform distribution* is defined by an equal value of a probability density function over a finite interval in $\mathbb{R}$. Thus, for $\Omega = [a, b]$ the uniform probability density function

[3] Many images in this document are currently taken from other sources. This is generally not a good practice, and we will be replacing these images someday soon. We want to highlight that we do not encourage this for formal documents, but only use them here in these educational notes for your benefit temporarily.

Figure 1.3: Selection of a random number (x) from the unit interval $[0, 1]$.

$\forall \omega \in [a, b]$ is defined as

$$p(\omega) = \frac{1}{b-a}.$$

One can also define Uniform(a, b) by taking $\Omega = \mathbb{R}$ and setting $p(\omega) = 0$ whenever ω is outside of $[a, b]$. This form is convenient because $\Omega = \mathbb{R}$ can then be used consistently for all one-dimensional probability distributions. When we do this, we will refer to the subset of $\mathbb{R}$ where $p(\omega) > 0$ as the*support* of the density function.

The *exponential distribution* is defined over a set of non-negative numbers; i.e., $\Omega = [0, \infty)$. For parameter $\lambda > 0$, its pdf is

$$p(\omega) = \lambda e^{-\lambda \omega}.$$

As the name suggests, this pdf has an exponential form, with sharply decreasing probability for values x as they increase in magnitude. As before, the sample space can be extended to all real numbers, in which case we would set $p(\omega) = 0$ for $\omega < 0$.

The *Gaussian distribution* or normal distribution is one of the most frequently used probability distributions. It is defined over $\Omega = \mathbb{R}$, with two parameters, $\mu \in \mathbb{R}$ and $\sigma > 0$ and pdf

$$p(\omega) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(\omega-\mu)^2}$$

As we will discuss next, for a random variable that is Gaussian distributed, the parameter μ is the mean or expected value and σ^2 is the variance. We will refer to this distribution as Gaussian(μ, σ^2) or $\mathcal{N}(\mu, \sigma^2)$. When the mean is zero, and the variance is 1 (unit variance), this Gaussian is called the *standard normal*; it has a name because it is so frequently used. Both Gaussian and exponential distribution are members of a broader family of distributions called the natural exponential family. We will see a general definition of this family later, in Section 7.2.

The *gamma distribution* is used to model waiting times, and is similar to the Poisson distribution but for continuous variables. It is defined over $\Omega = (0, \infty)$, with shape parameter $\alpha > 0$ and rate parameter $\beta > 0$ and pdf

$$p(\omega) = \frac{\beta^\alpha}{\Gamma(\alpha)} \omega^{\alpha-1} e^{-\beta\omega}$$

where $\Gamma(\alpha)$ is called the gamma function. A random variable that is gamma-distributed is denoted $X \sim \text{Gamma}(\alpha, \beta)$.

Example 5: Consider selecting a number (x) between 0 and 1 uniformly randomly (Figure 1.3). What is the probability that the number is greater than or equal to $\frac{3}{4}$ or less than and equal to $\frac{1}{4}$?

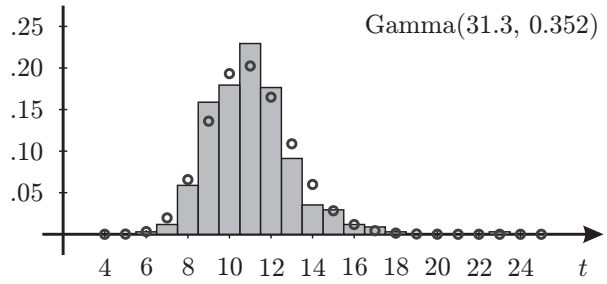


Figure 1.4: A histogram of recordings of the commute time (in minutes) to work. The data set contains 340 measurements collected over one year, for a distance of roughly 3.1 miles. The data was modeled using a gamma family of probability distributions, with the particular location and scale parameters estimated from the raw data. The values of the gamma distribution are shown as dark circles.

We know that $\Omega = [0, 1]$. The distribution is defined by the uniform pdf, $p(\omega) = \frac{1}{b-a} = 1$ where $a = 0, b = 1$ define the interval for the outcome space. We define the event of interest as $A = \left[0, \frac{1}{4}\right] \cup [\frac{3}{4}, 1]$ and calculate its probability as

$$\begin{aligned} P(A) &= \int_0^{1/4} p(\omega)d\omega + \int_{3/4}^{1} p(\omega)d\omega \qquad \triangleright\ p(\omega) = 1 \\ &= \left(\frac{1}{4} - 0\right) + \left(1 - \frac{3}{4}\right) \\ &= \frac{1}{2}. \end{aligned}$$

What if we had instead asked the probability that the number is *strictly greater* than $\frac{3}{4}$ or *strictly less* than $\frac{1}{4}$? Because the probability of any individual event in the continuous case is 0, there is no difference in integration if we consider open or closed intervals. Therefore, the probability would still be $\frac{1}{2}$. □

Example 6: Let's imagine you have collected your commute times for the year[4], and would like to model the probability of your commute time to help you can make predictions about your commute time tomorrow. For this setting, your random variable X corresponds to the commute time, and you need to define probabilities for this random variable. This data could be considered to be discrete, taking values, in minutes, $\{4, 5, 6, \ldots, 26\}$. You could then create histograms of this data (table of probability values), as shown in Figure 1.4, to reflect the likelihood of commute times.

The commute time, however, is not actually discrete, and so you would like to model it as a continuous RV. One reasonable choice is a gamma distribution. How, though, does one take the recorded data and determine the parameters α, β to the gamma distribution? Estimating these parameters is actually quite straightforward, though not as immediately obvious as estimating tables of probability values; we discuss how to do so in Chapter 3. The learned gamma distribution is also depicted in Figure 1.4. Given the gamma distribution, one could now ask the question: what is my expected commute time today? A natural

[4]as co-author Predrag amazingly did, and you get to see his fascinating data here.

answer is to actually take the expected value (mean) of this gamma distribution, which we define below in Section 1.4. □

1.3 Multivariate random variables

Much of the above development extends to multivariate random variables—a vector of random variables—because the definition of outcome spaces and probabilities is general. The examples so far, however, have dealt with scalar random variables, because for multivariate random variables, we need to understand how variables interact. In this section, we discuss several new notions that only arise when there are multiple random variables, including joint distributions, conditional distributions, marginals and dependence between variables.

Let us start with a simpler example, with two discrete random variables X and Y with outcome spaces $\mathcal{X}$ and $\mathcal{Y}$. There is a joint probability mass function $p : \mathcal{X} \times \mathcal{Y} \to [0, 1]$, and corresponding joint probability distribution P, such that

$$p(x, y) = P(X = x, Y = y)$$

where the pmf needs to satisfy

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) = 1.$$

For example, if $\mathcal{X} = \{\text{young}, \text{old}\}$ and $\mathcal{Y} = \{\text{no arthritis}, \text{arthritis}\}$, then the pmf could be the table of probabilities

		Y = 0	Y = 1
X	0	$1/2$	$1/100$
	1	$1/10$	$39/100$

This fits within the definition of probability spaces, because $\Omega = \mathcal{X} \times \mathcal{Y}$ is a valid space, and $\sum_{\omega \in \Omega} p(\omega) = \sum_{(x,y) \in \Omega} p(x, y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y)$. We consider the random variable $Z = (X, Y)$ to be a multivariate random variable, of two dimensions.

Looking at the joint probabilities in the table, we can definitely see that the two random variables interact. The joint probability is small for young and having arthritis, for example. Further, there seems to be more magnitude in the rows corresponding to young, suggesting that the probabilities are also influenced by the proportion of people in the population that are old or young. In fact, one might ask if we can figure out this proportion just from this table.

The answer is a resounding yes, and leads us to marginal distributions and why we might care about marginal distributions. Given a joint distribution over random variables, one would hope that we could extract more specific probabilities, like the distribution over just one of those variables, which is called the marginal distribution. The marginal can be simply computed, by summing up over all values of the other variable

$$P(X = \text{young}) = p(\text{young}, \text{no arthritis}) + p(\text{young}, \text{arthritis}) = \tfrac{51}{100}.$$

A young person either does or does not have arthritis, so summing up over these two possible cases factors out that variable. Therefore, using data collected for random variable

$Z = (X, Y)$, we can determine the proportion of the population that is young and the proportion that is old.

In general, we can consider d-dimensional random variable $\boldsymbol{X} = (X_1, X_2, \ldots, X_d)$ with vector-valued outcomes $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$, such that each x_i is chosen from some $\mathcal{X}_i$. Then, for the discrete case, any function $p : \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_d \to [0, 1]$ is called a multidimensional probability mass function if

$$\sum_{x_1 \in \mathcal{X}_1} \sum_{x_2 \in \mathcal{X}_2} \cdots \sum_{x_d \in \mathcal{X}d} p\left(x_1, x_2, \ldots, x_d\right) = 1.$$

or, for the continuous case, $p : \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_d \to [0, \infty]$ is a multidimensional probability density function if

$$\int_{\mathcal{X}_1} \int_{\mathcal{X}_2} \cdots \int_{\mathcal{X}d} p\left(x_1, x_2, \ldots, x_d\right) dx_1 dx_2 \ldots dx_d = 1.$$

A *marginal distribution* is defined for a subset of $\boldsymbol{X} = (X_1, X_2, \ldots, X_d)$ by summing or integrating over the remaining variables. For the discrete case, the marginal distribution $p\left(x_i\right)$ is defined as

$$p\left(x_i\right) = \sum_{x_1 \in \mathcal{X}_1} \cdots \sum_{x_{i-1} \in \mathcal{X}_{i-1}} \sum_{x_{i+1} \in \mathcal{X}_{i+1}} \cdots \sum_{x_d \in \mathcal{X}_d} p\left(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_d\right),$$

where the variable x_i is fixed to some value and we sum over all possible values of the other variables. Similarly, for the continuous case, the marginal distribution $p\left(x_i\right)$ is defined as

$$p\left(x_i\right) = \int_{\mathcal{X}_1} \cdots \int_{\mathcal{X}_{i-1}} \int_{\mathcal{X}_{i+1}} \cdots \int_{\mathcal{X}_d} p\left(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_d\right) dx_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_d.$$

Notice that we use p to define the density over $\boldsymbol{x}$, but then we overload this terminology and also use p for the density only over x_i. To be more precise, we should define two separate functions (pdfs), say $p_{\boldsymbol{x}}$ for the density over the multivariate random variable and p_{x_i} for the marginal. It is common, however, to simply use p, and infer the random variable from context. In most cases, it is clear; if it is not, we will explicitly highlight the pdfs with additional subscripts.

We can define common multivariate pmfs and pdfs, that are extensions of the scalar pmfs and pdfs. The most useful extensions include table of probability values, uniform distributions and Gaussian distributions. We define these concretely in the final section of this chapter, Section 1.5, for reference. Before we provide these definitions, however, it will be useful to understand how multiple variables interact, because this will influence the extension from univariate to multivariate. In particular, it will be useful to understand conditional distributions and dependence, which we discuss next.

1.3.1 Conditional distributions

Conditional probabilities define probabilities of a random variable X, given information about the value of another random variable Y. More formally, the conditional probability $p(y|x)$ for two random variables X and Y is defined as

$$p(y|x) = \frac{p(x, y)}{p(x)} \tag{1.1}$$

where $p(x) > 0$.

Exercise 2: Verify that $p(y|x)$ sums (integrates) to 1 over all values $y \in \mathcal{Y}$ for a fixed given $x \in \mathcal{X}$, and thus satisfies the conditions of a probability mass (density) function. □

Equation (1.1) now allows us to calculate the posterior probability of an event A, given some observation x, as

$$P(Y \in A|X = x) = \begin{cases} \sum_{y \in A} p(y|x) & Y : \text{discrete} \\ \\ \int_A p(y|x)dy & Y : \text{continuous} \end{cases}$$

Writing $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$ is called the *product rule*. The extension to more than two variables is straightforward. We can write

$$p(x_d|x_1, \ldots, x_{d-1}) = \frac{p(x_1, \ldots, x_d)}{p(x_1, \ldots, x_{d-1})}.$$

By a recursive application of the product rule, we obtain

$$p(x_1, \ldots, x_d) = p(x_1) \prod_{i=2}^{d} p(x_i|x_1, \ldots, x_{i-1}) \tag{1.2}$$

which is referred to as the *chain rule* or *general product rule*. For example, for three variables, the product rule gives

$$p(x_1, x_2, x_3) = p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$$

This rule also applies to collections of random variables, where a collection can be treated as one random variable. For example,

$$p(x_1, x_2, x_3) = p(x_2, x_3|x_1)p(x_1)$$

This arises because (x_2, x_3) have a valid probability space, so we can use the product rule for two variables: x_1 and (x_2, x_3). Using the product rule, giving $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$, we can also derive *Bayes' rule*:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{1.3}$$

Therefore, one really only needs to remember the product rule, to easily recall Bayes' rule.

You may notice that the order of variables in the product rule did not seem to matter. It is in fact somewhat interesting that we can either define the conditional distribution $p(x|y)$ and marginal $p(y)$ or we can define $p(y|x)$ and p(x) and both equivalently recover the joint distribution $p(x, y)$. This property is simply a fact of the definition of conditional distributions, and provides flexibility when estimating distributions. We will most use this equivalence in the form of Bayes rule, when doing parameter estimation and maximum likelihood. For work in graphical models, which is not discussed here, this flexibility is of even greater importance.

1.3.2 Independence of random variables

Two random variables are *independent* if their joint probability distribution factors into the product of the marginals

$$p(x, y) = p(x)p(y).$$

One intuitive reason for this definition can be seen by considering X conditioned on Y. If $p(x|y) = p(x)$, then this means that the value of Y has no influence on the distribution over X, and so they are independent. From the product rule, we know $p(x, y) = p(x|y)p(y)$ and since $p(x|y) = p(x)$, this gives $p(x, y) = p(x)p(y)$ as defined above.

The notion of independence can be generalized to than two random variables. More generally, d random variables are said to be *mutually independent* or *jointly independent* if a joint probability distribution of any subset of variables can be expressed as a product of marginal probability distributions of its components

$$p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2)\ldots p(x_d).$$

Another form of independence, called *conditional independence*, is used even more frequently in machine learning. It represents independence between variables in the presence of some other random variable (evidence); e.g.,

$$p(x, y|z) = p(x|z)p(y|z)$$

Interestingly, the two forms of independence are unrelated: neither one implies the other. X and Y can be independent, but not conditionally independent given Z. X and Y can be conditionally independent given Z, but not independent. We show this in two simple examples in Figure A.1 in the Appendix.

Here, we provide an example more directly related to machine learning, about why we care about independence and conditional independence. If two variables are independent, this has important modeling implications. For example, if feature X and target Y are independent, then X is not useful for predicting Y and so is not a useful feature. If two variables are conditionally independent given another variable, this can also have important modeling implications. For example, if we have two features X_1 and X_2, with target Y, if X_2 and Y are conditionally independent given X_1, then feature X_2 is redundant and could potentially be discarded. As a concrete example, let X_1 = temperature in Celcius and X_2 = temperature in Fahrenheit, with Y = plants need watering. Y is definitely not independent of X_2; however, once X_1 is known (or given), then there is no additional information to be gained from X_2 and so $p(y|x_1, x_2) = p(y|x_1)$. In general, recognizing independencies and conditional independencies can inform and simplify the modeling procedure, and we will see several examples in terms of simplifying maximum likelihood for i.i.d. data and in naive Bayes for classification. We end this section with one more example, using a biased coin, to highlight the distinction between independence and conditional independence.

Example 7: [Biased coin and conditional independence] Assume a manufacturer has produced a biased coin, where it does not equally randomly give heads (H) or tails (T). Rather, it actually has some unknown probability α of seeing H when flipping the coin. Because this bias is unknown, we will encode our uncertainty by defining a random Z = bias of the coin. In general, this random variable can take values in $[0, 1]$. For the purposes of

this example, let's make this a bit simpler, and assume that we know the bias is one of $\mathcal{Z} = \{0.1, 0.5, 0.8\}$. If the bias is 0.5, that would mean this is an unbiased (fair) coin. Let's further assume that we think the probability is equally likely, meaning $P(Z = z) = 1/3$, because the manufacturer gave us no reason to think any of 0.1, 0.5 or 0.8 to be more likely.

Now imagine that you flip the coin twice, and record the two outcomes x_1 and x_2. These two separate flips correspond to two random variables, X_1 and X_2. The outcome space for X_1 and X_2 is $\{\mathrm{H}, \mathrm{T}\}$ with true distribution $P(X_i = \mathrm{H}|\text{bias} = \alpha) = \alpha$. Since we do not know the bias α, and have chosen to treat the bias as a random variable Z, what we actually know is that $P(X_i|Z = z) = z$, i.e., X_i is a Bernoulli random variable with parameter z. Given information about the bias, we can specify the distribution over X_i. Otherwise, without knowing the bias, the distribution over X_1 is the marginal, with $x \in \{\mathrm{H}, \mathrm{T}\}$

$$\begin{aligned}
P(X_1 = x) &= \sum_{z \in \mathcal{Z}} P(X_1 = x, Z = z) \\
&= \sum_{z \in \mathcal{Z}} P(X_1 = x|Z = z)P(Z = z) && \triangleright \text{ product rule} \\
&= P(X_1 = x|Z = 0.1)P(Z = 0.1) + P(X_1 = x|Z = 0.5)P(Z = 0.5) + \\
&\quad + P(X_1 = x|Z = 0.8)P(Z = 0.8)
\end{aligned}$$

Are X_1 and X_2 conditionally independent given Z? The answer is yes, because given the bias of the coin, knowing the outcome of X_2 does not influence the distribution over X_1, i.e.,

$$P(X_1 = x_1, X_2 = x_2|Z = z) = P(X_1 = x_1|Z = z)P(X_2 = x_2|Z = z)$$

Regardless of what we observe for X_2, we know the distribution over X_1 is a Bernoulli with the given bias z.

Are X_1 and X_2 independent? The answer is no, because without knowing the bias of the coin, knowing the outcome of X_2 tells us something about the likely Bernoulli distribution over X_1. For example, if $X_1 = \mathrm{T}$ and $X_2 = \mathrm{H}$, then the second outcome suggests that the bias might not be totally skewed towards T. More formally,

$$\begin{aligned}
P(X_1 = x_1, X_2 = x_2) &= \sum_{z \in \mathcal{Z}} P(X_1 = x_1, X_2 = x_2|Z = z)P(Z = z) \\
&= \sum_{z \in \mathcal{Z}} P(X_1 = x_1|Z = z)P(X_2 = x_2|Z = z)P(Z = z)
\end{aligned}$$

which is not guaranteed to equal $P(X_1 = x_1)P(X_2 = x_2)$, where

$$\begin{aligned}
&P(X_1 = x_1)P(X_2 = x_2) \\
&= \left(\sum_{z_1 \in \mathcal{Z}} P(X_1 = x_1|Z = z_1)P(Z = z_1) \right) \left(\sum_{z_2 \in \mathcal{Z}} P(X_2 = x_2|Z = z_2)P(Z = z_2) \right)
\end{aligned}$$

□

1.4 Expectations and moments

The *expected value*, or *mean*, of a random variable X is the average of repeatedly sampled x, in the limit of sampling. It is not necessarily the value we expect to see most frequently—that is called the mode. More precisely, given the pmf or pdf p for outcome space $\mathcal{X}$, the expectation of X is

$$\mathbb{E}[X] = \begin{cases} \sum_{x \in \mathcal{X}} xp(x) & \text{if } X \text{ is discrete} \\ \\ \int_{\mathcal{X}} xp(x)dx & \text{if } X \text{ is continuous} \end{cases}$$

For a dice roll, where each number from 1 to 6 has uniform probability, the expected value is 3.5 and the mode is tied for all numbers (i.e., it is multi-modal). For a Bernoulli distribution, where $\mathcal{X} = \{0, 1\}$, the expected value is α, which is not even an outcome that will be observed, but is the average of 0s and 1s if we flipped the coin infinitely many times. The mode in this case depends on α: if $\alpha > 0.5$, making 1 have higher probability, then the mode is 1; if $\alpha < 0.5$, the mode is 0; otherwise, it is bimodal with modes 0 and 1. For a Gaussian distribution, the expected value is the parameter μ, and the mode also equals μ.

In general, we may be interested in the expected value of functions of the random variable X. For example, we may want to know $\mathbb{E}[X^2]$, or more generally $\mathbb{E}\left[X^k\right]$ for some $k > 1$. Or, we may want to know $\mathbb{E}\left[(X - c)^k\right]$ for some $k > 1$ and a constant c. These are called the *moments* of X. In general, for a function $f : \mathcal{X} \to \mathbb{R}$, we can consider $f(X)$ to be a transformed random variables and define its expectation as

$$\mathbb{E}[f(X)] = \begin{cases} \sum_{x \in \mathcal{X}} f(x)p(x) & \text{if } X \text{ is discrete} \\ \\ \int_{\mathcal{X}} f(x)p(x)dx & \text{if } X \text{ is continuous} \end{cases}$$

If $\mathbb{E}[f(X)] = \pm\infty$, we say that the expectation does not exist or is not well-defined.

One useful moment is the *variance*: the central second moment[5], where central indicates $c = \mathbb{E}[X]$. The variance indicates the amount that the random variable varies around its mean. For example, for a Gaussian distribution, if the variance σ^2 is large, then the Gaussian is very wide, indicating a non-negligible density for a broader range of points x around μ. Alternatively, if σ^2 is almost zero, then the Gaussian is concentrated tightly around μ.

We can also consider conditional expectations, and expectations for multivariate random variables. For two random variables X and Y and function $f : \mathcal{Y} \to \mathbb{R}$, the conditional expectation is

$$\mathbb{E}[f(Y)|x] = \begin{cases} \sum_{y \in \mathcal{Y}} f(y)p(y|x) & \text{if } Y \text{ is discrete} \\ \\ \int_{\mathcal{Y}} f(y)p(y|x)dy & \text{if } Y \text{ is continuous} \end{cases}$$

Using the identity function $f(y) = y$ results in the standard conditional expectation $\mathbb{E}[Y|x]$. We shall see later that under some conditions $\mathbb{E}[Y|x]$ is referred to as the *regression function*.

[5]The expected value is typically called the *first moment* of X, with c assumed to be zero, because $\mathbb{E}[X - c] = \mathbb{E}[X] - c$, so it is not particularly interesting to include c in the expectation.

The reason for this is that $\mathbb{E}[Y|x]$ is a function of x; if x is a feature, then the expected value of Y (the target) given x is a reasonable prediction.

Exercise 3: Show the *law of total expectations*: $\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y|X]]$, where the outer expectation is over X and the inner expectation is over Y. For example, if both Y and X are discrete

$$\begin{aligned}\mathbb{E}[\mathbb{E}[Y|X]] &= \sum_{x\in\mathcal{X}} p(x)\mathbb{E}[Y|X=x] \\ &= \sum_{x\in\mathcal{X}} p(x) \sum_{y\in\mathcal{Y}} yp(y|x)\end{aligned}$$

□

For two random variables X and Y and $f : \mathcal{X}\times\mathcal{Y} \to \mathbb{R}$, we can also define the expectation over the joint distribution, with one variable fixed

$$\mathbb{E}[f(X,y)] = \begin{cases} \sum_{x\in\mathcal{X}} f(x,y)p(x) & \text{if } X \text{ is discrete} \\ \\ \int_{\mathcal{X}} f(x,y)p(x)dx & \text{if } X \text{ is continuous} \end{cases}$$

or over both variables

$$\mathbb{E}[f(X,Y)] = \begin{cases} \sum_{y\in\mathcal{Y}} \mathbb{E}[f(X,y)|y]p(y) & \text{if } Y \text{ is discrete} \\ \\ \int_{\mathcal{Y}} \mathbb{E}[f(X,y)|y]p(y)dy & \text{if } Y \text{ is continuous} \end{cases}$$

For example, if X is continuous and Y is discrete, this gives

$$\mathbb{E}[f(X,Y)] = \int_{\mathcal{X}} \left(\sum_{y\in\mathcal{Y}} f(x,y)p(x,y) \right) dx$$

Notice that this matches the above, because $p(x,y) = p(y|x)p(x)$.

Exercise 4: Show that $\int_{\mathcal{X}} \left(\sum_{y\in\mathcal{Y}} f(x,y)p(x,y) \right) dx = \int_{\mathcal{X}} \mathbb{E}[f(x,Y)|x]p(x)dx$. □

Just as above with variance, the *covariance* is one important instance of these expected values, with $f(x,y) = (x - \mathbb{E}[X])(y - \mathbb{E}[Y])$. The expected value under this function indicates how the two variable vary together. We use specific notation for the covariance, because it is so frequently used

$$\begin{aligned}\mathrm{Cov}[X,Y] &= \mathbb{E}\left[(X - \mathbb{E}[X])\,(Y - \mathbb{E}[Y])\right] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]\,,\end{aligned}$$

with $\mathrm{Cov}[X,X] = V[X]$ being the variance of the random variable X. The *correlation* is the covariance, normalized by the standard deviation—square root of the variance—of each random variable

$$\mathrm{Corr}[X,Y] = \frac{\mathrm{Cov}[X,Y]}{\sqrt{V[X]}\sqrt{V[Y]}}.$$

The covariance can become larger, if X and Y themselves have large variance. The correlation, on the other hand, is guaranteed to be between -1 and 1, and so is an scale-invariant measure of how the variables vary together.

In many situations we need to analyze more than two random variables. A simple two-dimensional summary of all pairwise covariance values involving d random variables $X_1, X_2, \ldots, X_d$ is called the covariance matrix. The covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$ has the (i, j) entry defined as

$$\begin{aligned}\Sigma_{ij} &= \mathrm{Cov}[X_i, X_j] \\ &= \mathbb{E}\left[(X_i - \mathbb{E}\left[X_i\right])\left(X_j - \mathbb{E}\left[X_j\right]\right)\right]\end{aligned}$$

with the full matrix written as

$$\begin{aligned}\mathbf{\Sigma} &= \mathrm{Cov}[\boldsymbol{X}, \boldsymbol{X}] \\ &= \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^\top] \\ &= \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^\top] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^\top.\end{aligned}$$

The second line involves the *outer product* of the vector $\mathbf{v} = \boldsymbol{X} - \mathbb{E}[\boldsymbol{X}] \in \mathbb{R}^d$, to produce $\mathbf{A} = \mathbf{v}\mathbf{v}^\top \in \mathbb{R}^{d \times d}$. This outer product is a matrix multiplication, where the first matrix is $d \times 1$ and the second is $1 \times d$. Using the rules of matrix multiplication, this gives $\mathbf{A}_{i,j} = \mathbf{v}_i \mathbf{v}_j$. Notice that the diagonal elements of the $d \times d$ covariance matrix are the variances for each variable X_i and the off-diagonal elements are the covariance values between pairs of variables. The covariance matrix is symmetric and positive semi-definite, i.e., $\mathbf{\Sigma} \succeq 0$. We will discuss more about positive semi-definite matrices later in the notes.

Properties of expectations

Here we review some useful properties of expectations. We can generically consider multivariate random variables, $\boldsymbol{X} \in \mathbb{R}^d$ and $\boldsymbol{Y} \in \mathbb{R}^m$, for $d, m \in \mathbb{N}$, with univariate random variables as a special case. We consider the more general case because it will be useful to start thinking directly in terms of random vectors. For a constant $c \in \mathbb{R}$, it holds that:

1. $\mathbb{E}\left[c\boldsymbol{X}\right] = c\mathbb{E}\left[\boldsymbol{X}\right]$
2. $\mathbb{E}\left[\boldsymbol{X} + \boldsymbol{Y}\right] = \mathbb{E}\left[\boldsymbol{X}\right] + \mathbb{E}\left[\boldsymbol{Y}\right]$ ▷ when $d = m$
3. $V\left[c\right] = 0$ ▷ the variance of a constant is zero
4. $\mathrm{V}[\boldsymbol{X}] = \mathrm{Cov}[\boldsymbol{X}, \boldsymbol{X}] \succeq 0$ (i.e., is positive semi-definite), where for $d = 1$, $\mathrm{V}[\boldsymbol{X}] \geq 0$ is a scalar. We use $\mathrm{V}[\boldsymbol{X}]$ as a shorthand for $\mathrm{Cov}[\boldsymbol{X}, \boldsymbol{X}]$.
5. $\mathrm{V}[c\boldsymbol{X}] = c^2\mathrm{V}[\boldsymbol{X}]$.
6. $\mathrm{Cov}[\boldsymbol{X}, \boldsymbol{Y}] = \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])(\boldsymbol{Y} - \mathbb{E}(\boldsymbol{Y})^\top] = \mathbb{E}[\boldsymbol{X}\boldsymbol{Y}^\top] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{Y}]^\top$
7. $\mathrm{V}[\boldsymbol{X} + \boldsymbol{Y}] = \mathrm{V}[\boldsymbol{X}] + \mathrm{V}[\boldsymbol{Y}] + 2\mathrm{Cov}[\boldsymbol{X}, \boldsymbol{Y}]$ ▷ when $d = m$
8. $\mathrm{Cov}[\boldsymbol{X}_1 + \boldsymbol{X}_2 + \ldots + \boldsymbol{X}_m] = \sum_{i=1}^{m}\sum_{j=1}^{m}\mathrm{Cov}[\boldsymbol{X}_i, \boldsymbol{X}_j] = \sum_{i=1}^{m}\mathrm{V}[\boldsymbol{X}_i] + 2\sum_{1 \leq i < j \leq m}\mathrm{Cov}[\boldsymbol{X}_i, \boldsymbol{X}_j]$

In addition, if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent random variables, it holds that:

1. $\mathbb{E}[X_i Y_j] = \mathbb{E}[X_i]\,\mathbb{E}[Y_j]$ for all i, j
2. $\mathrm{V}[\boldsymbol{X} + \boldsymbol{Y}] = \mathrm{V}[\boldsymbol{X}] + \mathrm{V}[\boldsymbol{Y}]$ ▷ when $d = m$
3. $\mathrm{Cov}[\boldsymbol{X}, \boldsymbol{Y}] = 0$.

1.5 Multivariate PMFs and PDFs

We can now consider extensions to definitions of pmfs and pdfs, for the multivariate setting.

The *multivariate Gaussian distribution* is a generalization of the Gaussian or normal distribution to the d-dimensional case, with $\Omega = \mathbb{R}^d$. It is defined as

$$p(\boldsymbol{\omega}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{\omega} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\omega} - \boldsymbol{\mu})\right),$$

with parameters $\boldsymbol{\mu} \in \mathbb{R}^d$ and positive-definite matrix $\boldsymbol{\Sigma}$, which is the covariance matrix. The term $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$. We will refer to this distribution as Gaussian$(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ or $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For the multivariate case, to define the Gaussian, we need to consider how the variables vary together. For example, if the variables are independent, then the covariance matrix is diagonal and the Gaussian is nicely spherical (if both variables have unit variance). If the variables are not independent, the Gaussian becomes sheared, skewed away from its mean differently than can be accounted for solely by the variables variance.

One example of a multidimensional pmf is the *multinomial distribution*, which is used to model a sequence of n independent and identically distributed (i.i.d.) trials with d outcomes. At each point $(k_1, k_2, \ldots, k_d)$ in the sample space, the multinomial pmf provides the probability that the outcome 1 occurred k_1 times, outcome 2 occurred k_2 times, etc. Of course, $0 \leq k_i \leq n$ for $\forall i$ and $\sum_{i=1}^d k_i = n$. The outcome space is $\Omega = \{0, 1, \ldots, n\}^d$ and the multinomial pmf is defined as

$$p(k_1, k_2, \ldots, k_d) = \begin{cases} \binom{n}{k_1, k_2, \ldots, k_d} \alpha_1^{k_1} \alpha_2^{k_2} \ldots \alpha_d^{k_d} & k_1 + k_2 + \cdots + k_d = n \\ \\ 0 & \text{otherwise} \end{cases}$$

where α_i's are positive coefficients such that $\sum_{i=1}^d \alpha_i = 1$. That is, each coefficient α_i gives the probability of outcome i in any trial. The multinomial coefficient

$$\binom{n}{k_1, k_2, \ldots, k_d} = \frac{n!}{k_1! k_2! \cdots k_d!}$$

generalizes the binomial coefficient by enumerating all ways in which one can distribute n balls into d boxes such that the first box contains k_1 balls, the second box k_2 balls, etc.

Consider an example with a fair six-sided dice. A multinomial distribution well describes an experiment consisting of n tosses of this dice and counting the number of occurrences of each number. In this setting, $d = 6$, $\alpha_i = 1/6$, for each $i \in \{1, 2, 3, 4, 5, 6\}$. More directly

related to machine learning, we will see the multinomial for multi-class classification, where d is the number of classes and $n = 1$.

Example 8: [Curse of dimensionality for discrete multidimensional random variables] One way to avoid pdfs is to discretize variables, and then define a pmf that is a table of probability values. Though reasonable in some cases, in general this can exponentially increase the number of parameters for the probability distribution and is one example of the *curse of dimensionality*.

To see why, consider the following example. Assume we have a d-dimensional random variable, with each entry taking values between 0 and 1 (i.e., $\mathcal{X}_i = [0, 1]$). You decide to discretize this should that each entry is put into one of three bins, changing $\mathcal{X}_i = \{1, 2, 3\}$. Now you have a lot of flexibility in specifying these probabilities in your pmf, unlike say a Gaussian that has a more rigid functional form. Unfortunately, however, this table of values could be very large, with 3^d entries. This requires you to specify $3^d - 1$ probability values, where one of the values is automatically set to one minus the sum of all the other probabilities to ensure you have a valid pmf.

If, instead, you had specified a Gaussian distribution on these variables, the number of parameters needed to define the distribution is only $d + d^2$, which is likely to be much less. □

Chapter 2

Introduction to Optimization

Much of machine learning deals with learning functions by finding the optimal function according to an objective. For example, one may be interested in finding a function that minimizes the squared differences to some targets for all the samples: $\sum_{i=1}^{n}(f(\mathbf{x}_i) - y_i)^2$. To find such a function, you need to have a basic grasp of optimization techniques.

In this chapter, we discuss basic optimization tools, for generic smooth objectives. Many of the algorithms in machine learning rely on a simple approach: gradient descent. We first discuss how to minimize objectives using both first and second-order gradient descent. This overview covers only a small part of optimization, but fortunately, many machine learning algorithms are based on these simple optimization approaches. We provide more optimization background later, in Chapter 6, once you have had a chance to use this more basic optimization knowledge in the next couple of chapters.

2.1 The basic optimization problem and stationary points

A basic optimization goal is to select a set of parameters $\mathbf{w} \in \mathbb{R}^d$ to minimize a given objective function $c : \mathbb{R}^d \to \mathbb{R}$

$$\min_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w})$$

For example, to obtain the parameters $\mathbf{w}$ for linear regression that minimizes the squared differences, we use $c(\mathbf{w}) = \sum_{i=1}^{n}(\langle \mathbf{x}_i, \mathbf{w}\rangle - y_i)^2$, for dot product

$$\langle \mathbf{x}_i, \mathbf{w}\rangle = \sum_{j=1}^{d} x_{ij}, w_j.$$

We use the term objective here, rather than error, since error has an explicit connotation that the function is inaccurate. Later we will see that objectives will include both error terms— indicating how accurately they recreate data—as well as terms the provide other preferences on the function. Combining these terms with the error produces the final objective we would like to minimize. For example, for linear regression, we will optimize a regularized objective, $c(\mathbf{w}) = \sum_{i=1}^{n}(\langle \mathbf{x}_i, \mathbf{w}\rangle - y_i)^2 + \sum_{j=1}^{d} w_j^2$ where the second term encodes a preference for smaller coefficients w_j.

The goal then is to find $\mathbf{w}$ that minimizes the objective. The most straightforward, naive solution could be to do a random search: generate random $\mathbf{w}$ and check $c(\mathbf{w})$. If any newly generated $\mathbf{w}_t$ on iteration t outperforms the previous solution best solution $\mathbf{w}$, in that $c(\mathbf{w}_t) < c(\mathbf{w})$, then we can set $\mathbf{w}_t$ to be the new optimal solution. We will assume that our objectives are continuous, and so can take advantage of this smoothness to design better search strategies. In particular, for smooth functions, we will be able to use gradient descent, which we describe in the next section.

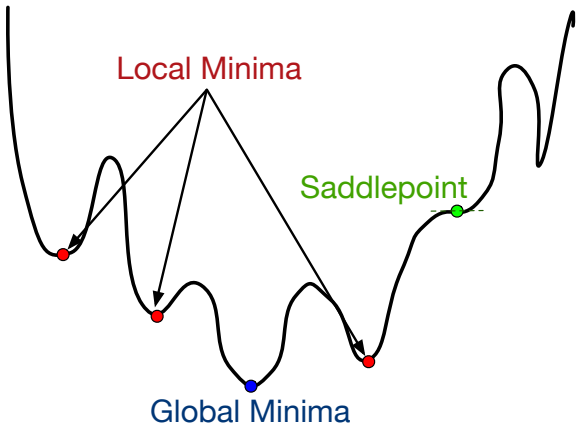


Figure 2.1: Stationary points on a smooth function surface: local minima, global minima and saddlepoints.

Gradient descent enables us to reach stationary points: points $\mathbf{w}$ where the gradient is zero. Consider first the univariate case. The derivative tells us the rate of change of the function surface at a point w. When the derivative of the objective is zero at $w \in \mathbb{R}$, i.e., $\frac{d}{dw}c(w) = 0$, this means that locally the function surface is flat. Such points correspond to local minima, local maxima and saddlepoints, as show in Figure 2.1. For example, assume again that we are doing linear regression, with only one feature and so only one weight $w \in \mathbb{R}$. The derivative of the objective $c(w) = \sum_{i=1}^{n}(x_i w - y_i)^2$ is

$$\begin{aligned}\frac{d}{dw}c(w) &= \frac{d}{dw}\sum_{i=1}^{n}(x_i w - y_i)^2 \\ &= \sum_{i=1}^{n}\frac{d}{dw}(x_i w - y_i)^2 \\ &= \sum_{i=1}^{n} 2(x_i w - y_i)x_i\end{aligned}$$

where the last step follows from the chain rule. Our goal is to find w such that $\frac{d}{dw}c(w) = 0$; once we find such a stationary point, we can then determine if it is a local minimum, local maximum or saddlepoint. Because this objective is convex, we in fact know that all stationary points must be global minima, and so we would not need to do this check. We discuss this further in the last section, where we discuss some properties of objectives.

For the multivariate case, we need to consider gradients instead of derivatives. For $\mathbf{w} \in \mathbb{R}^d$ where $d > 1$, we need to ask: how does the function change locally, depending on how each element of $\mathbf{w}$ is changed? To quantify this, we use the gradient which is composed of partial derivatives

$$\nabla c(\mathbf{w}) = \left[\frac{\partial c}{\partial w_1}(\mathbf{w}) \quad \frac{\partial c}{\partial w_2}(\mathbf{w}) \quad \ldots \quad \frac{\partial c}{\partial w_d}(\mathbf{w})\right].$$

Each partial derivative $\frac{\partial c}{\partial w_j}(\mathbf{w})$ represents how the function c changes, when only w_j is changed and the other $w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_d$ are kept constant. For example, for $c(\mathbf{w} =$

$(w_1, w_2)) = \frac{1}{2}(x_1 w_1 + x_2 w_2 - y)^2$, the partial derivatives are

$$\frac{\partial c}{\partial w_1}(\mathbf{w}) = (x_1 w_1 + x_2 w_2 - y)x_1$$
$$\frac{\partial c}{\partial w_2}(\mathbf{w}) = (x_1 w_1 + x_2 w_2 - y)x_2$$

Usefully, we do not have to consider how the whole vector changes jointly in all the variables. Rather, it is sufficient to find stationary points by finding $\mathbf{w}$ where the partial derivatives are zero.

2.2 Gradient descent

The key idea behind gradient descent is to approximate the function with a Taylor series approximation. This approximation facilitates computation of a descent direction locally on the function surface. We begin by considering the univariate setting, with $w \in \mathbb{R}$. A function $c(w)$ in the neighborhood of point w_0, can be approximated using the Taylor series as

$$c(w) = \sum_{n=0}^{\infty} \frac{c^{(n)}(w_0)}{n!}(w - w_0)^n,$$

where $c^{(n)}(w_0)$ is the n-th derivative of function $c(w)$ evaluated at point w_0. This assumes that $c(w)$ is be infinitely differentiable, but in practice we will take such polynomial approximations for a finite n. A *second-order* approximation to this function uses the first three terms of the series as

$$c(w) \approx \hat{c}(w) = c(w_0) + (w - w_0)c'(w_0) + \frac{1}{2}(w - w_0)^2 c''(w_0).$$

A stationary point of this $\hat{c}(w)$ can be easily found by finding the first derivative and setting it to zero

$$c'(w) \approx c'(w_0) + (w - w_0)c''(w_0) = 0.$$

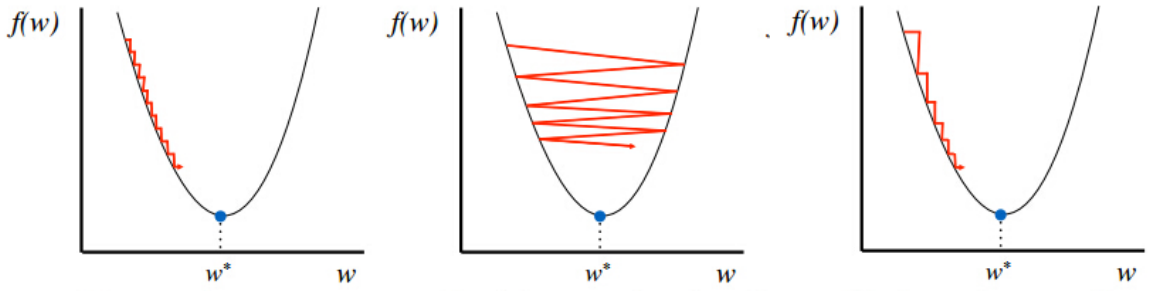
Solving this equation for w gives us

$$w_1 = w_0 - \frac{c'(w_0)}{c''(w_0)}.$$

Locally, this new w_1 will be an improvement on w_0, and will be a stationary point of this local approximation $\hat{c}$. Moving (far enough) from w_0, however, makes this local second-order Taylor series inaccurate. We would need to check the local approximation at this new point w_1, to determine if we can further improve locally. Therefore, to find the optimal w, we can iteratively apply this procedure

$$w_{t+1} = w_t - \frac{c'(w_t)}{c''(w_t)}. \tag{2.1}$$

constantly improving w_i until we reach a point where the derivative is zero, or nearly zero. This method is called the Newton-Raphson method, or second-order gradient descent.



(a) Step-size too small (b) Step-size too big (c) Adaptive step-size

Figure 2.2: Different optimization paths, due to different stepsize choices.

In first-order gradient descent, the approximation is worse, where we no longer use the second derivative. Instead, when taking a first-order approximation, we know that we are ignoring $O((w - w_0)^2)$ terms, and so the local approximation becomes

$$c(w) \approx \hat{c}(w) = c(w_0) + (w - w_0)c'(w_0) + \frac{1}{2\eta}(w - w_0)^2$$

for some constant $\frac{1}{\eta}$ reflecting the magnitude of the ignored $O((w - w_0)^2)$ terms. The resulting update is then, for step-size η_t

$$w_{t+1} = w_t - \eta_t c'(w_t). \tag{2.2}$$

From this, one can see that, given access to the second derivative, a reasonable choice for the stepsize is $\eta_t = \frac{1}{c''(w_t)}$.

We can similarly obtain such rules for multivariate variables. For example, gradient descent for $c : \mathbb{R}^d \to \mathbb{R}$ consists of the update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla c(\mathbf{w}_t). \tag{2.3}$$

where

$$\nabla c(\mathbf{w}_t) = \left(\frac{\partial c}{\partial w_1}(\mathbf{w}_i), \frac{\partial c}{\partial w_2}(\mathbf{w}_t), ..., \frac{\partial c}{\partial w_d}(\mathbf{w}_t) \right) \qquad \in \mathbb{R}^d$$

is the gradient of function c evaluated at $\mathbf{w}_t$. We will discuss how to derive this update in the multivariable setting in Chapter 6.

2.3 Selecting the step-size

An important part of (first-order) gradient descent is to select the step-size. If the step-size is too small, then many iterations are required to reach a stationary point (Figure 2.2(a)); If the step-size is too large, then you are likely to oscillate around the minimum (Figure 2.2 (b)). What we really want is an adaptive step-size (Figure 2.2 (c)), that likely starts larger and then slowly reduces over time as a stationary point is approached.

The basic method to obtain adaptive step-sizes is to use *line search*. The idea springs from the following goal: we would like to obtain the optimal step-size according to

$$\min_{\eta \in \mathbb{R}^+} c(\mathbf{w}_t - \eta \nabla c(\mathbf{w}_t))$$

The solution to this optimization corresponds to the best scalar stepsize we could select, for the current point $\mathbf{w}_t$ with descent direction $-\nabla c(\mathbf{w}_t)$. Solving this optimization would be too expensive; however, we can find approximate solutions quickly. One natural choice is to use a backtracking line search, that tries the largest reasonable stepsize $\eta_{\max}$, and then reduces it until the objective is decreased. The idea is to search along the line of possible $\eta \in (0, \eta_{\max}]$, with the intuition that a large step is good—as long as it does not overshoot. If it does overshoot, then the stepsize was too large, and should be reduced. The reduction is typically according to the rule $\tau\eta$ for some $\tau \in [0.5, 0.9]$. For $\tau = 0.5$, the stepsize reduces more quickly—halves on each step of the backtracking line search; for $\tau = 0.9$, the search more slowly backtracks from $\eta_{\max}$. As soon as a stepsize is found that decreases the objective, it is accepted. We then obtain a new $\mathbf{w}_t$, again compute the gradient and start the line search once again from $\eta_{\max}$.

One can imagine better strategies for selecting the stepsize than this simplistic search; we will in fact discuss some of these in Section 6.5. Nonetheless, this basic line search—an improvement therein—remains a common approach for selecting the stepsize.

Algorithm 1: Line Search($\mathbf{w}_t, c, \mathbf{g} = \nabla c(\mathbf{w}_t)$)

Optimization parameters: $\eta_{\max} = 1.0, \tau = 0.7,$ tolerance $\leftarrow 10e^{-4}$
$\eta \leftarrow \eta_{\max}$
$\mathbf{w} \leftarrow \mathbf{w}_t$
obj $\leftarrow c(\mathbf{w})$
while number of backtracking iterations is less than maximum iterations **do**
    $\mathbf{w} \leftarrow \mathbf{w}_t - \eta\mathbf{g}$
    // Ensure improvement is at least as much as tolerance
    If $c(\mathbf{w}) <$ obj - tolerance **then break**
    // Else, the objective is worse and so we decrease stepsize
    $\eta \leftarrow \tau\eta$
if maximum number of iterations reached **then**
    // Could not improve solution
    return $\mathbf{w}_t, \eta = 0$
return $\mathbf{w}, \eta$

2.4 Optimization properties

There are several optimization properties to keep in mind when reading this handbook, which we highlight here.

Maximizing versus minimizing We have so far discussed the goal of minimizing an objective; an equivalent alternative is to maximize the negative of this objective.

$$\operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) = \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^d} -c(\mathbf{w})$$

where argmin returns $\mathbf{w}$ that produces the minimum value of $c(\mathbf{w})$ and argmax returns $\mathbf{w}$ that produces the maximum value of $-c(\mathbf{w})$. The actual min and max values are not

the same, since for a given optimal solution, $c(\mathbf{w}) \neq -c(\mathbf{w})$. We opt to formulate each of our optimizations as a minimization, and do gradient descent. It would be equally valid, however, to formulate the optimizations as maximizations, and do gradient ascent.

Convexity A function $c : \mathbb{R}^d \to \mathbb{R}$ is said to be convex if for any $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ and $t \in [0, 1]$,

$$c(t\mathbf{w}_1 + (1-t)\mathbf{w}_2) \leq tc(\mathbf{w}_1) + (1-t)c(\mathbf{w}_2) \tag{2.4}$$

This definition means that when we draw a line between any two points on the function surface, the function values between these two points all lie below this line. Convexity is an important property, because it means that every stationary point is a global minimum. Therefore, regardless of where we start our gradient descent, with appropriately chosen stepsize and sufficient iterations, we will reach an optimal solution.

A corresponding definition is a concave function, which is precisely the opposite: all points lie above the line. For any convex function c, the negative of that function $-c$ is a concave function.

Uniqueness of the solution We often care if there is more than one solution to our optimization problem. In some cases, we care about *identifiability*, which means we can identify the true solution. If there is more than one solution, one might consider that the problem is not precisely posed. For some problems, it is important or even necessary to have identifiability (e.g., estimating the percentage of people with a disease) whereas for other we simply care about finding a suitable (predictive) function f that reasonably accurately predicts the targets, even if it is not the unique such function. We will not consider identifiability further in this document, but it is important to be cognizant of if your objective has multiple solutions.

Equivalence under a constant shift Adding or multiplying by a constant $a \neq 0$ does not change the solution

$$\operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} a\ c(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) + a.$$

You can see why by taking the gradient of all three objectives and noticing that the gradient is zero under the same conditions

$$\nabla a\ c(\mathbf{w}) = 0 \iff a\nabla c(\mathbf{w}) = 0 \iff \nabla c(\mathbf{w}) = 0$$

and

$$\nabla(c(\mathbf{w}) + a) = 0 \iff \nabla c(\mathbf{w}) = 0.$$

Chapter 3

Basic Principles of Parameter Estimation

In probabilistic modeling, we are typically presented with a set of observations and the objective is to find a model, or function, $\hat{f}$ that shows good agreement with the data and respects certain additional requirements. We shall roughly categorize these requirements into three groups: (*i*) the ability to generalize well, (*ii*) the ability to incorporate prior knowledge and assumptions into modeling, and (*iii*) scalability. First, the model should be able to stand the test of time; that is, its performance on the previously unseen data should not deteriorate once this new data is presented. Models with such performance are said to generalize well. Second, $\hat{f}$ must be able to incorporate information about the model space $\mathcal{F}$ from which it is selected and the process of selecting a model should be able to accept training "advice" from an analyst. Finally, when large amounts of data are available, learning algorithms must be able to provide solutions in reasonable time given the resources such as memory or CPU power. In summary, the choice of a model ultimately depends on the observations at hand, our experience with modeling real-life phenomena, and the ability of algorithms to find good solutions given limited resources.

An easy way to think about finding the "best" model is through learning parameters of a distribution. Suppose we are given a set of observations $\mathcal{D} = \{x_i\}_{i=1}^n$, where $x_i \in \mathbb{R}$ and have knowledge that $\mathcal{F}$ is a family of all univariate Gaussian distributions; e.g., $\mathcal{F} = \text{Gaussian}(\mu, \sigma^2)$, with $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$. In this case, the problem of finding the best model (by which we mean function) can be seen as finding the best parameters μ^* and σ^*; i.e., the problem can be seen as *parameter estimation*. We call this process estimation because the typical assumption is that the data was generated by an unknown model from $\mathcal{F}$ whose parameters we are trying to recover from data.

We will formalize parameter estimation using probabilistic techniques and will subsequently find solutions through optimization, occasionally with constrains in the parameter space. The main assumption throughout this part will be that the set of observations $\mathcal{D}$ was generated (or collected) independently and according to the same distribution $p(x)$. The statistical framework for model inference is shown in Figure 3.1.

3.1 Maximum a posteriori and maximum likelihood estimation

The idea behind *maximum a posteriori* (MAP) estimation is to find the most probable model for the observed data. Given the data set $\mathcal{D}$, we formalize the MAP solution as

$$f_{\text{MAP}} = \arg\max_{f \in \mathcal{F}} \{p(f|\mathcal{D})\},$$

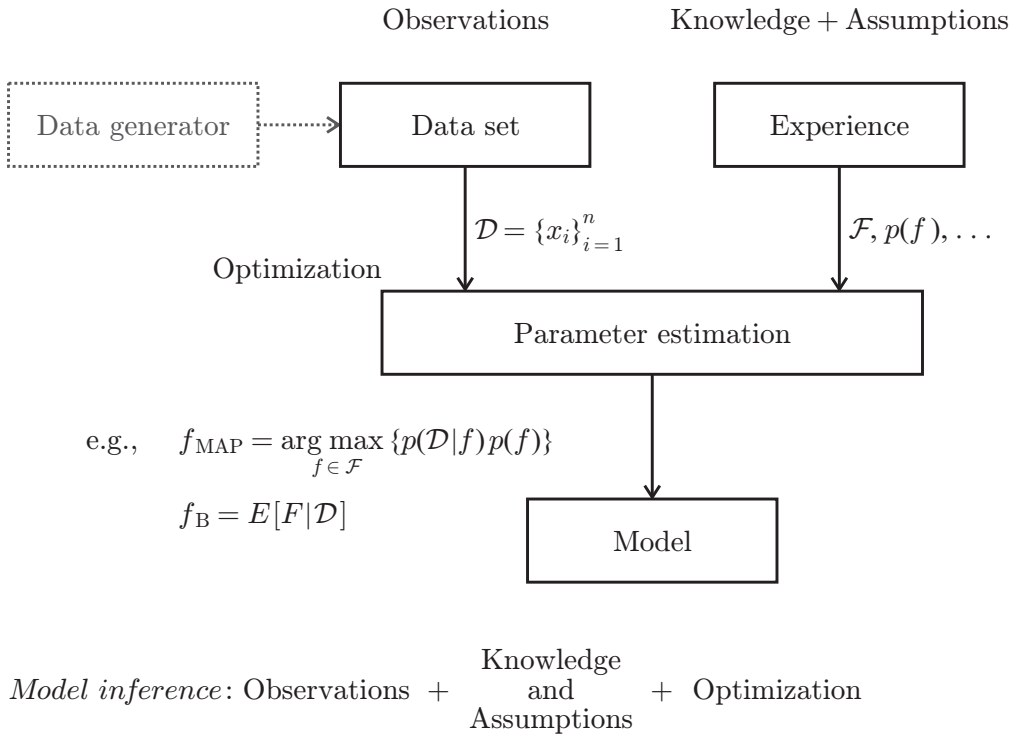


Figure 3.1: Statistical framework for model inference. The estimates of the parameters are made using a set of observations $\mathcal{D}$ as well as experience in the form of model space $\mathcal{F}$, prior distribution $p(f)$, or specific starting solutions in the optimization step.

where $p(f|\mathcal{D})$ is called the *posterior distribution* of the model given the data. In discrete model spaces, $p(f|\mathcal{D})$ is the probability mass function and the MAP estimate is exactly the most probable model. Its counterpart in continuous spaces is the model with the largest value of the posterior density function. Note that we use words *model*, which is a function, and its *parameters*, which are the coefficients of that function, somewhat interchangeably. However, we should keep in mind the difference, even if only for pedantic reasons.

To calculate the posterior distribution we start by applying the Bayes rule as

$$p(f|\mathcal{D}) = \frac{p(\mathcal{D}|f) \cdot p(f)}{p(\mathcal{D})}, \tag{3.1}$$

where $p(\mathcal{D}|f)$ is called the *likelihood* function, $p(f)$ is the *prior* distribution of the model, and $p(\mathcal{D})$ is the *marginal* distribution of the data. Notice that we use $\mathcal{D}$ for the observed data set, but that we usually think of it as a realization of a multidimensional random variable D drawn according to some distribution $p(\mathcal{D})$. Using the formula of total probability, we can express $p(\mathcal{D})$ as

$$p(\mathcal{D}) = \begin{cases} \sum_{f \in \mathcal{F}} p(\mathcal{D}|f)p(f) & f : \text{discrete} \\ \\ \int_{\mathcal{F}} p(\mathcal{D}|f)p(f)df & f : \text{continuous} \end{cases}$$

Therefore, the posterior distribution can be fully described using the likelihood and the prior. The field of research and practice involving ways to determine this distribution and

optimal models is referred to as *inferential statistics.* The posterior distribution is sometimes referred to as inverse probability.

Finding f_{MAP} can be greatly simplified because $p(\mathcal{D})$ in the denominator does not affect the solution. We shall re-write Equation (3.1) as

$$\begin{aligned} p(f|\mathcal{D}) &= \frac{p(\mathcal{D}|f) \cdot p(f)}{p(\mathcal{D})} \\ &\propto p(\mathcal{D}|f) \cdot p(f), \end{aligned}$$

where $\propto$ is the proportionality symbol. Thus, we can find the MAP solution by solving the following optimization problem

$$f_{\text{MAP}} = \arg\max_{f \in \mathcal{F}} \{p(\mathcal{D}|f)p(f)\}.$$

In some situations we may not have a reason to prefer one model over another and can think of $p(f)$ as a constant over the model space $\mathcal{F}$. Then, maximum a posteriori estimation reduces to the maximization of the likelihood function; i.e.,

$$f_{\text{ML}} = \arg\max_{f \in \mathcal{F}} \{p(\mathcal{D}|f)\}.$$

We will refer to this solution as the *maximum likelihood* solution. Formally speaking, the assumption that $p(f)$ is constant is problematic because a uniform distribution cannot be always defined (say, over $\mathbb{R}$), though there are some solutions to this issue using improper priors. Nonetheless, it may be useful to think of the maximum likelihood approach as a separate technique, rather than a special case of MAP estimation, but keep this connection in mind.

Observe that MAP and ML approaches report solutions corresponding to the mode of the posterior distribution and the likelihood function, respectively. We shall later contrast this estimation technique with the view of the Bayesian statistics in which the goal is to minimize the posterior risk. Such estimation typically results in calculating conditional expectations, which can be complex integration problems. From a different point of view, MAP and ML estimates are called *point estimates*, as opposed to estimates that report confidence intervals for a particular group of parameters.

Example 9: Suppose data set $\mathcal{D} = \{2, 5, 9, 5, 4, 8\}$ is an i.i.d. sample from a Poisson distribution with a fixed but unknown parameter λ_0. Find the maximum likelihood estimate of λ_0.

The probability density function of a Poisson distribution is expressed as $p(x|\lambda) = \lambda^x e^{-\lambda}/x!$, with some parameter $\lambda \in \mathbb{R}^+$. We will estimate this parameter as

$$\lambda_{\text{ML}} = \arg\max_{\lambda \in (0,\infty)} \{p(\mathcal{D}|\lambda)\}. \tag{3.2}$$

We can write the likelihood function as

$$\begin{aligned} p(\mathcal{D}|\lambda) &= p(\{x_i\}_{i=1}^n |\lambda) \\ &= \prod_{i=1}^n p(x_i|\lambda) \\ &= \frac{\lambda^{\sum_{i=1}^n x_i} \cdot e^{-n\lambda}}{\prod_{i=1}^n x_i!}. \end{aligned}$$

To find λ that maximizes the likelihood, we will first take a logarithm (a monotonic function) to simplify the calculation, then find its first derivative with respect to λ, and finally equate it with zero to find the maximum. Specifically, we express the log-likelihood $ll(D, \lambda) = \ln p(\mathcal{D}|\lambda)$ as

$$ll(\mathcal{D}, \lambda) = \ln \lambda \sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n} \ln (x_i!)$$

and proceed with the first derivative as

$$\begin{aligned} \frac{\partial ll(\mathcal{D}, \lambda)}{\partial \lambda} &= \frac{1}{\lambda} \sum_{i=1}^{n} x_i - n \\ &= 0. \end{aligned}$$

By substituting $n = 6$ and values from $\mathcal{D}$, we can compute the solution as

$$\begin{aligned} \lambda_{\text{ML}} &= \frac{1}{n} \sum_{i=1}^{n} x_i \\ &= 5.5, \end{aligned}$$

which is simply a sample mean. The second derivative of the likelihood function is always negative because λ must be positive; thus, the previous expression indeed maximizes the likelihood. Note that to properly maximize this loss, we also need to ensure the constraint $\lambda \in (0, \infty)$ is enforced. Because the solution above is in the constraint set, we know we have the correct solution to Equation (3.2); however, in other situations, we will have to explicitly enforce constraints in the optimization, as we will discuss later.

□

Example 10: Let $\mathcal{D} = \{2, 5, 9, 5, 4, 8\}$ again be an i.i.d. sample from Poisson(λ_0), but now we are also given additional information. Suppose the prior knowledge about λ_0 can be expressed using a gamma distribution $\Gamma(x|k, \theta)$ with parameters $k = 3$ and $\theta = 1$. Find the maximum a posteriori estimate of λ_0.

First, we write the probability density function of the gamma family as

$$\Gamma(x|k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)},$$

where $x > 0$, $k > 0$, and $\theta > 0$. $\Gamma(k)$ is the gamma function that generalizes the factorial function; when k is an integer, we have $\Gamma(k) = (k-1)!$. The MAP estimate of the parameters can be found as

$$\lambda_{\text{MAP}} = \underset{\lambda \in (0,\infty)}{\arg\max} \left\{ p(\mathcal{D}|\lambda) p(\lambda) \right\}.$$

As before, we can write the likelihood function as

$$p(\mathcal{D}|\lambda) = \frac{\lambda^{\sum_{i=1}^{n} x_i} \cdot e^{-n\lambda}}{\prod_{i=1}^{n} x_i!}$$

and the prior distribution as

$$p(\lambda) = \frac{\lambda^{k-1} e^{-\frac{\lambda}{\theta}}}{\theta^k \Gamma(k)}.$$

Now, we can maximize the logarithm of the posterior distribution $p(\lambda|\mathcal{D})$ using

$$\begin{aligned}\ln p(\lambda|\mathcal{D}) &\propto \ln p(\mathcal{D}|\lambda) + \ln p(\lambda) \\ &= \ln \lambda(k - 1 + \sum_{i=1}^{n} x_i) - \lambda(n + \frac{1}{\theta}) - \sum_{i=1}^{n} \ln x_i! - k \ln \theta - \ln \Gamma(k)\end{aligned}$$

to obtain

$$\begin{aligned}\lambda_{\text{MAP}} &= \frac{k - 1 + \sum_{i=1}^{n} x_i}{n + \frac{1}{\theta}} \\ &= 5\end{aligned}$$

after incorporating all data.

A quick look at λ_{MAP} and λ_{ML} suggests that as n grows, both numerators and denominators in the expressions above become increasingly more similar. In fact, it is a well-known result that, in the limit of infinite samples, both the MAP and ML converge to the same model, f, as long as the prior does not have zero probability on f. This result shows that the MAP estimate approaches the ML solution for large data sets. In other words, large data diminishes the importance of prior knowledge. This is an important conclusion because it simplifies mathematical apparatus necessary for practical inference.

To get some intuition for this result, we will show that the MAP and ML estimates converge to the same solution for the above example with a Poisson distribution. Let $s_n = \sum_{i=1}^{n} x_i$, which is a sample from the random variable $S_n = \sum_{i=1}^{n} X_i$. If $\lim_{n\to\infty} s_n/n^2 = 0$ (i.e., s_n does not grow faster than n^2), then

$$\begin{aligned}|\lambda_{\text{MAP}} - \lambda_{\text{ML}}| &= \left| \frac{k - 1 + s_n}{n + 1/\theta} - \frac{s_n}{n} \right| \\ &= \left| \frac{k - 1}{n + 1/\theta} - \frac{s_n}{n(n + 1/\theta)} \right| \\ &\leq \frac{|k - 1|}{n + 1/\theta} + \frac{s_n}{n(n + 1/\theta)} \qquad \xrightarrow[n\to\infty]{} 0\end{aligned}$$

Note that if $\lim_{n\to\infty} s_n/n^2 \neq 0$, then both estimators go to ∞; however, such a sequence of values has an essentially zero probability of occurring. Consistency theorems for ML and MAP estimation state that convergence to the true parameters occurs "almost surely" or "with probability 1" to indicate that these unbounded sequences constitute a set of measure-zero, under certain reasonable conditions (for more, see [19, Theorem 9.13]).

□

Example 11: Let $\mathcal{D} = \{x_i\}_{i=1}^{n}$ be an i.i.d. sample from a univariate Gaussian distribution. Find the maximum likelihood estimates of the parameters.

We start by forming the log-likelihood function

$$\begin{aligned}\ln p(\mathcal{D}|\mu,\sigma) &= \ln \prod_{i=1}^{n} p(x_i|\mu,\sigma) \\ &= n \ln \frac{1}{\sqrt{2\pi}} + n \ln \frac{1}{\sigma} - \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{2\sigma^2}.\end{aligned}$$

We now compute the partial derivatives of the log-likelihood with respect to all parameters as

$$\frac{\partial}{\partial \mu} \ln p(\mathcal{D}|\mu,\sigma) = \frac{\sum_{i=1}^{n} (x_i - \mu)}{\sigma^2}$$

and

$$\frac{\partial}{\partial \sigma} \ln p(\mathcal{D}|\mu,\sigma) = -\frac{n}{\sigma} + \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{\sigma^3}.$$

From here, we can proceed to derive that

$$\mu_{\text{ML}} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

and

$$\sigma^2_{\text{ML}} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_{\text{ML}})^2 .$$

□

3.2 Maximum likelihood for conditional distributions

We can also formulate maximum likelihood problems for conditional distributions. Recall that a conditional distribution has the form $p(y|x)$, for two random variables Y and X, where above we considered the marginal distribution $p(x)$ or $p(y)$. For the distributions above, we asked: what is the distribution over this variable? For a conditional distribution, we are instead asking: given some auxiliary information, now what is the distribution over this variable? When the auxiliary information changes, so will the distribution over the variable. For example, we may want to condition a distribution over sales of a particular product (Y) given the current month (X). We expect the distribution over Y to be different, depending on the month.

Conditional distributions can be from any of the distribution families discussed above, and we can similarly formulate parameter estimation problems. The parameters, however, are usually tied to the given variable X. We provide a simple example to demonstrate this below. Much of the parameter estimation formulations we consider in the remainder of the book will be for conditional distributions, because in machine learning we typically have a large number of auxiliary variables (features) and are trying to predict (or learn the distribution over) targets. In the chapters on regression and classification, we will

demonstrate how many models can be formulated as maximum likelihood for conditional distributions $p(y|\mathbf{x})$.

Example 12: Assume you are given two random variables X and Y and that you believe $p(y|x) = \mathcal{N}(\mu = x, \sigma^2)$ for some unknown σ. Our goal is to estimate this unknown parameter σ. Notice that the distribution over Y varies, depending on which X value is observed or given.

We again start by forming the log-likelihood function, now for pairs of n samples $\mathcal{D} = (x_1, y_1), \ldots, (x_n, y_n)$. We will use the chain rule: $p(x_i, y_i) = p(y_i|x_i)p(x_i)$.

$$\begin{aligned}
\ln p(\mathcal{D}|\sigma) &= \ln \prod_{i=1}^{n} p(x_i, y_i|\sigma) \\
&= \ln \prod_{i=1}^{n} p(y_i|x_i, \sigma)p(x_i) \\
&= n \ln \frac{1}{\sqrt{2\pi}} + n \ln \frac{1}{\sigma} - \frac{\sum_{i=1}^{n} (y_i - x_i)^2}{2\sigma^2} + \sum_{i=1}^{n} \ln p(x_i).
\end{aligned}$$

Notice that the last line uses $\mu = x_i$ for each normal distribution $p(y_i|x_i, \sigma)$. We now compute the partial derivatives of the log-likelihood with respect to the parameter σ

$$\frac{\partial}{\partial \sigma} \ln p(\mathcal{D}|\sigma) = -\frac{n}{\sigma} + \frac{\sum_{i=1}^{n} (y_i - x_i)^2}{\sigma^3}.$$

Notice that $\frac{\partial}{\partial \sigma} \sum_{i=1}^{n} \ln p(x_i) = 0$, because σ does not parameterize $p(x_i)$. Therefore, to obtain the optimal σ, we do not need to know or specify the distribution over the random variable X. By setting the derivative to zero, to obtain a stationary point, we obtain

$$\sigma^2_{\mathrm{ML}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i)^2.$$

□

3.3 [Advanced] The relationship between maximizing likelihood and Kullback-Leibler divergence

We now investigate the relationship between maximum likelihood estimation and Kullback-Leibler divergence. Kullback-Leibler divergence between two probability distributions $p(x)$ and $q(x)$ is defined on $\mathcal{X} = \mathbb{R}$ as

$$D_{\mathrm{KL}}(p||q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx.$$

In information theory, Kullback-Leibler divergence has a natural interpretation of the inefficiency of signal compression when the code is constructed using a suboptimal distribution $q(x)$ instead of the correct (but unknown) distribution $p(x)$ according to which the data has been generated. However, more often than not, Kullback-Leibler divergence is simply considered to be a measure of divergence between two probability distributions. Although this

divergence is not a metric (it is not symmetric and does not satisfy the triangle inequality) it has important theoretical properties in that (i) it is always non-negative and (ii) it is equal to zero if and only if $p(x) = q(x)$.

Consider now a divergence between an estimated probability distribution $p(x|\theta)$ and an underlying (true) distribution $p(x|\theta_0)$ according to which the data set $\mathcal{D} = \{x_i\}_{i=1}^n$ was generated. The Kullback-Leibler divergence between $p(x|\theta)$ and $p(x|\theta_0)$ is

$$\begin{aligned} D_{\mathrm{KL}}(p(x|\theta_0)||p(x|\theta)) &= \int_{-\infty}^{\infty} p(x|\theta_0) \log \frac{p(x|\theta_0)}{p(x|\theta)} dx \\ &= \int_{-\infty}^{\infty} p(x|\theta_0) \log \frac{1}{p(x|\theta)} dx - \int_{-\infty}^{\infty} p(x|\theta_0) \log \frac{1}{p(x|\theta_0)} dx. \end{aligned}$$

The second term in the above equation is simply the (differential) entropy of the true distribution and is not influenced by our choice of the model θ. The first term, on the other hand, can be expressed as

$$\int_{-\infty}^{\infty} p(x|\theta_0) \log \frac{1}{p(x|\theta)} dx = -E[\log p(X|\theta)]$$

Therefore, maximizing $E[\log p(X|\theta)]$ minimizes the Kullback-Leibler divergence between $p(x|\theta)$ and $p(x|\theta_0)$. Using the strong law of large numbers, we know that

$$\frac{1}{n} \sum_{i=1}^{n} \log p(x_i|\theta) \quad \overset{a.s.}{\to} \quad E[\log p(X|\theta)]$$

when $n \to \infty$. Thus, when the data set is sufficiently large, maximizing the likelihood function minimizes the Kullback-Leibler divergence and leads to the conclusion that $p(x|\theta_{\mathrm{ML}}) = p(x|\theta_0)$, if the underlying assumptions are satisfied. Under reasonable conditions, we can infer from it that $\theta_{\mathrm{ML}} = \theta_0$. This will hold for families of distributions for which a set of parameters uniquely determines the probability distribution; e.g., it will not generally hold for mixtures of distributions but we will discuss this situation later. This result is only one of the many connections between statistics and information theory.

Chapter 4

Introduction to Prediction Problems

Machine learning addresses many problem settings, which can sometimes feel overwhelming. As a non-exhaustive list, these include supervised learning (with classification and regression); semi-supervised learning; unsupervised learning; completion under missing features; structured prediction; learning to rank; statistical relational learning; active learning; and temporal prediction (with time series prediction and policy evaluation in reinforcement learning and online learning). For some of these settings, such as active learning and reinforcement learning, the data collection is a central part of the algorithm and can significantly determine the quality of the learned predictive models. Most other settings assume that data has been collected—without our ability to influence that collection—and now we simply need to analyze that data and learn the best predictors that we can. In this passive setting, we can either assume that the data is i.i.d.—which is the most common—or that there are dependencies between data points—such as in time series prediction or statistical relational learning. There are also settings where the data is incomplete, say because a user did not fill in their age.

One ontology, therefore, could consider the following dimensions to categorize machine learning problems: passive vs. active; i.i.d. vs. non-i.i.d.; complete vs. incomplete. As with all ontologies, each problem will not perfectly fit into these categories. Further, it is likely that most data collection is not completely passive (even if only because the human modeler influences collection of data), is likely not i.i.d. (even if we intended it to be), and likely has some missing components. Nonetheless, algorithms will make these assumptions, to varying degrees, even if the data does not satisfy those assumptions. For the majority of these notes, we will focus on the simplest setting: passive, i.i.d. and complete. Further, because so many techniques arise from the basic classification and regression models within supervised learning, we will focus mostly on these.

In this chapter, we will first introduce classification and regression and then discuss criteria for selecting functions for classification and regression, to motivate the algorithms developed in later chapters.

4.1 Supervised learning problems

We start by defining a data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathcal{X}$ is the i-th object and $y_i \in \mathcal{Y}$ is the corresponding target designation. We usually assume that $\mathcal{X} = \mathbb{R}^d$, in which case $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ is a d-dimensional vector called a *data point* or *instance*. Each dimension of x_j is typically called a *feature* or an *attribute*. We will often organize the dataset into a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where each row corresponds to a sample and each column corresponds to a feature (see Figure 4.1).

We generally distinguish between two related but different types of prediction problems:

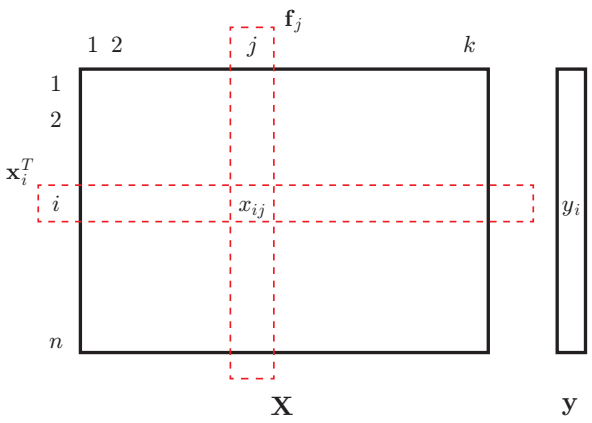


Figure 4.1: Data set representation and notation. $\mathbf{x}$ *is an* n*-by-*d *matrix representing features and data points, whereas* y *is an* n*-by-*1 *vector of targets.*

classification and regression. Generally speaking, we have a regression problem when $\mathcal{Y}$ is continuous and a classification problem if $\mathcal{Y}$ is discrete. Additionally, we might specify this distinction in the solution approach, i.e., using a regression approach (logistic regression) or a classification approach (naive Bayes). In both prediction scenarios, we assume that the features are easy to collect for each object (e.g., by measuring the height of a person or the square footage of a house), while the target variable is difficult to observe or expensive to collect (e.g., final selling price of a house). Such situations usually benefit from the construction of a computational model that predicts targets from a set of input values. The model is trained using a set of input objects for which target values have already been collected.

In regression we construct a function to predict continuous targets, such as $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = [0, \infty)$. For example, linear regression produces predictions in $\mathbb{R}$, and Poisson regression produces only positive predictions, i.e., in $[0, \infty)$. An example of a regression problem is shown in Table 4.1.

	size [sqft]	age [yr]	dist [mi]	inc [$]	dens [ppl/mi^2]	y
$\mathbf{x}_1$	1250	5	2.85	56,650	12.5	2.35
$\mathbf{x}_2$	3200	9	8.21	245,800	3.1	3.95
$\mathbf{x}_3$	825	12	0.34	61,050	112.5	5.10

Table 4.1: An example of a regression problem: prediction of the price of a house in a particular region. Here, features indicate the size of the house (size) in square feet, the age of the house (age) in years, the distance from the city center (dist) in miles, the average income in a one square mile radius (inc), and the population density in the same area (dens). The target indicates the price a house is sold at, e.g. in hundreds of thousands of dollars.

In classification we construct a function that predicts discrete class labels; this function is typically called a *classifier*. The cardinality of $\mathcal{Y}$ in classification problems is usually small, e.g. $\mathcal{Y} = \{\text{healthy}, \text{diseased}\}$. An example of a data set for classification with $n = 3$ data points and $d = 5$ features is shown in Table 4.2.

Classification problems can be further subdivided into multi-class and multi-label problems. A multi-class problem consists of providing the single label for an input. For example, for (simple) blood-type with $\mathcal{Y} = \{\text{A}, \text{B}, \text{AB}, \text{O}\}$, a patient can only be labeled with one of these labels. Within multi-class problems, if there are only two classes, it is called binary classification, such as the example in Table 4.2. In multi-label, an input can be associated with more than one label. An example of a multi-label problem is the classification of text documents into categories such as $\{\text{sports}, \text{medicine}, \text{travel}, \text{politics}\}$. Here, a single document may be related to more than one value in the set; e.g. an article on sports medicine. The learned function can now return multiple outputs.

Typically, to make the outputs more consistent between these two settings, the output for both multi-class and multi-label is an indicator vector. For $m = |\mathcal{Y}|$, the prediction for blood types might be [0 1 0 0] to indicate blood-type B and the prediction for four article labels could be [1 1 0 0] if it is both an article pertaining to sports and medicine.

Structured-output prediction $\mathcal{Y}$ can be a set of structured outputs, e.g. strings, trees, or graphs. This classification scenario is usually referred to as structured-output learning. The cardinality of the output space in structured-output learning problems is often very high. For example, when predicting the functions of a protein, an entire ontology tree needs to be predicted, as certain functionality is a subset of other functionality.

Deciding how to formalize the problem Observe that there does not exist a strict distinction between classification and regression. For example, consider the output space $\mathcal{Y} = \{0, 1, 2\}$. We can treat this as a multi-class classification problem, or we could presume $\mathcal{Y} = [0, 2]$ and learning a regression model. We can then threshold the predictions returned by the regression model, by rounding them to the closest integer.

How do you decide which problem formulation to use? Though the mathematical procedures in machine learning are precise, deciding how to formulate real-world problem is subtle, and so inherently less clear-cut. The selection of a particular way of modeling depends on the analyst and their knowledge of the domain as well as technical aspects of learning. In this example, you could ask: is there inherently an ordering to the outputs $\{0, 1, 2\}$? If not, say they correspond to Prefers apples, Prefers oranges, Prefers bananas, then it may be a poor choice to model the output as an interval, which often implies ordering. On the other hand, regression functions can be easier to learn and often produce surprisingly good classification predictions. Further, if there is an ordering to these class, say Good, Better, Best, then most classification models— which do not assume an ordering on the outputs—would not be able to take advantage of this ordering to improve prediction performance.

Selecting the function class, and objective, just like selecting distributions and priors, is an important step in using machine learning effectively. Fortunately, there is a wealth of knowledge, especially empirically, that can guide this selection. As you learn more about the methods, combined with some information about structure in your domain, you will become better at specifying the model class.

	wt [kg]	ht [m]	T [°C]	sbp [mmHg]	dbp [mmHg]	y
$\mathbf{x}_1$	91	1.85	36.6	121	75	-1
$\mathbf{x}_2$	75	1.80	37.4	128	85	$+1$
$\mathbf{x}_3$	54	1.56	36.6	110	62	-1

Table 4.2: An example of a binary classification problem: prediction of a disease state for a patient. Here, features indicate weight (wt), height (ht), temperature (T), systolic blood pressure (sbp), and diastolic blood pressure (dbp). The class labels indicate presence of a particular disease, e.g. diabetes. This data set contains one positive data point ($\mathbf{x}_2$) and two negative data points ($\mathbf{x}_1$, $\mathbf{x}_3$). The class label shows a disease state, i.e. $y_i = +1$ indicates the presence while $y_i = -1$ indicates absence of disease.

4.2 Unsupervised learning and semi-supervised learning

Datasets are not always complete: in some cases, we can only get labels for a small subset of instances, or we cannot get any labels at all. For example, when predicting whether a cat is in an image or not, we would need a human to take each image and label it with a 0 or 1. This labeling can be expensive, and so we can only expect that a small number of all pictures with cats have such an associated label. Using supervised learning only on this labeled subset if likely to produce a poor predictor, because of limited data. Semi-supervised learning deals with taking advantage of all the unlabeled data, to supplement the small labeled dataset, by finding structure in the features. For example, the features may lie on a lower-dimensional manifold; this structure could be inferred from the unlabeled data, and potentially more effectively restrict the function class to learn on the labeled dataset. *Unsupervised learning* is focused only on obtaining this structure, without the goal of learning a function to predict targets, because no targets are provided. Unsupervised learning will be discussed in Section 9.2.2, as a part of representation learning.

These two problem settings can be seen as an instance of a larger setting of learning under missing data. In general, it may not only be difficult to gather the outputs, but also some of the features. For example, when collecting patient data, it is likely that some patients will omit some information. Even though it is difficult to gather the information “has disease", it can also be difficult to ensure that other more straightforward data like “age" or “weight" is gathered. Further, one might even ask why there is a distinction between features and targets: they are all associated information about one item, like a patient. Given that a patient does have a disease, you may want to use this feature and their age to predict their weight—which they so blithely chose not to disclose. This more general way of approaching the problem can be useful when data is missing and leads to the general problem of completion. Different techniques are often used in such a setting, and we will not address it further until Section 9.2.2. For now, we keep the focus on supervised learning, which we will still be able to use even when some features are missing, using some simple heuristics for dealing with this missing data.

4.3 Optimal classification and regression models

Our goal now is to establish the performance criteria that will be used to evaluate predictors $f : \mathcal{X} \to \mathcal{Y}$ and subsequently define optimal classification and regression models. We start with classification and consider a situation where the joint probability distribution $p(\mathbf{x}, y)$ is either known or can be learned from data. Also, we will consider that we are given a cost function $\text{cost} : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, where for each prediction $\hat{y}$ and true target value y the classification cost can be expressed as a constant $\text{cost}(\hat{y}, y)$, regardless of the input $\mathbf{x} \in \mathcal{X}$ given to the classifier. This cost function can simply be stored as a $|\mathcal{Y}| \times |\mathcal{Y}|$ cost matrix.

The criterion for optimality of a classifier will be probabilistic. In particular, we are interested in minimizing the expected cost, for random variable $C = \text{cost}(f(X), Y)$,

$$\begin{aligned}\mathbb{E}[C] &= \int_{\mathcal{X}} \sum_{y} \text{cost}(\hat{y}, y) p(\mathbf{x}, y) d\mathbf{x} \\ &= \int_{\mathcal{X}} p(\mathbf{x}) \sum_{y} \text{cost}(\hat{y}, y) p(y|\mathbf{x}) d\mathbf{x},\end{aligned}$$

where the integration is over the entire input space $\mathcal{X} = \mathbb{R}^d$. From this equation, we can see that the optimal classifier can be expressed as

$$f_{\text{BR}}(\mathbf{x}) = \arg\min_{\hat{y} \in \mathcal{Y}} \left\{ \sum_{y} \text{cost}(\hat{y}, y) p(y|\mathbf{x}) \right\},$$

for any $\mathbf{x} \in \mathcal{X}$. We will refer to this classifier as the *Bayes risk classifier.* One example where the Bayes risk classifier can be useful is the medical domain. Suppose our goal is to decide whether a patient with a particular set of symptoms ($\mathbf{x}$) should be sent for an additional lab test ($y = 1$ if yes and $y = -1$ if not), with cost c_{lab}, in order to improve diagnosis. However, if we do not perform a lab test and the patient is later found to have needed the test for proper treatment, we may incur a significant penalty, say c_{lawsuit}. If $c_{\text{lawsuit}} \gg c_{\text{lab}}$, as it is expected to be, then the classifier needs to appropriately adjust its outputs to account for the cost disparity in different forms of incorrect prediction.

In many practical situations, however, it may not be possible to define a meaningful cost matrix and, thus, a reasonable criterion would be to minimize the probability of a classifier's error $P(f(\mathbf{x}) \neq y)$. This corresponds to the situation where the cost function is defined as

$$\text{cost}(\hat{y}, y) = \begin{cases} 0 & \text{when } y = \hat{y} \\ \\ 1 & \text{when } y \neq \hat{y} \end{cases}$$

After plugging these values in the definition for $f_{\text{BR}}(\mathbf{x})$, the Bayes risk classifier simply becomes the maximum a posteriori (MAP) classifier. That is,

$$f_{\text{MAP}}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \{p(y|\mathbf{x})\}.$$

Therefore, if $p(y|\mathbf{x})$ is known or can be accurately learned, we are fully equipped to make the prediction that minimizes the total cost. In other words, we have converted the problem

of minimizing the expected classification cost or probability of error, into the problem of learning functions, more specifically learning probability distributions.

The analysis for regression is a natural extension of that for classification. Here too, we are interested in minimizing the expected cost of prediction of the true target y when a predictor $f(\mathbf{x})$ is used. The expected cost can be expressed as

$$\mathbb{E}[C] = \int_{\mathcal{X}} \int_{\mathcal{Y}} \text{cost}(f(\mathbf{x}), y) p(\mathbf{x}, y) dy d\mathbf{x},$$

where $c : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ is again some cost function between the predicted value $f(\mathbf{x})$ and the true value y. For simplicity, we will consider

$$\text{cost}(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2,$$

which results in

$$\begin{aligned}
\mathbb{E}[C] &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(\mathbf{x}) - y)^2 p(\mathbf{x}, y) dy d\mathbf{x} \\
&= \int_{\mathcal{X}} p(\mathbf{x}) \underbrace{\int_{\mathcal{Y}} (f(\mathbf{x}) - y)^2 p(y|\mathbf{x}) dy}_{g(f(\mathbf{x}))} d\mathbf{x}.
\end{aligned}$$

Assuming $f(\mathbf{x})$ is flexible enough to be separately optimized for each unit volume $d\mathbf{x}$, we see that minimizing $\mathbb{E}[C]$ leads us to the problem of minimizing

$$g(u) = \int_{\mathcal{Y}} (u - y)^2 p(y|\mathbf{x}) dy,$$

where we used a substitution $u = f(\mathbf{x})$. We can now differentiate g with respect to u as

$$\begin{aligned}
&\frac{\partial g(u)}{\partial u} = 2 \int_{\mathcal{Y}} (u - y) p(y|\mathbf{x}) dy = 0 \\
&\implies u \underbrace{\int_{\mathcal{Y}} p(y|\mathbf{x}) dy}_{=1} = \int_{\mathcal{Y}} y p(y|\mathbf{x}) dy
\end{aligned}$$

which results in the optimal solution

$$\begin{aligned}
f^*(\mathbf{x}) &= \int_{\mathcal{Y}} y p(y|\mathbf{x}) dy \\
&= \mathbb{E}[Y|\mathbf{x}].
\end{aligned}$$

Therefore, the optimal regression model in the sense of minimizing the square error between the prediction and the true target is the conditional expectation $\mathbb{E}[Y|\mathbf{x}]$. It may appear that in the above equations, setting $f(\mathbf{x}) = y$ would always lead to $\mathbb{E}[C] = 0$. Unfortunately, this would be an invalid operation because for a single input $\mathbf{x}$ there may be multiple possible outputs y and they can certainly appear in the same data set. To be a well-defined function, $f(\mathbf{x})$ must always have the same output for the same input. $\mathbb{E}[C] = 0$ can only be achieved if $p(y|\mathbf{x})$ is a delta function for every $\mathbf{x}$.

Having found the optimal regression model, we can now write the expected cost in the cases of both optimal and suboptimal models $f(\mathbf{x})$. That is, we are interested in expressing $\mathbb{E}[C]$ when

1. $f(\mathbf{x}) = \mathbb{E}[Y|\mathbf{x}]$
2. $f(\mathbf{x}) \neq \mathbb{E}[Y|\mathbf{x}]$.

We have already found $\mathbb{E}[C]$ when $f(\mathbf{x}) = \mathbb{E}[Y|\mathbf{x}]$. The expected cost can be simply expressed as

$$\mathbb{E}[C] = \int_{\mathcal{X}} \int_{\mathcal{Y}} (\mathbb{E}[Y|\mathbf{x}] - y)^2 p(\mathbf{x}, y) dy d\mathbf{x},$$

which corresponds to the (weighted) squared error between the optimal prediction and the target everywhere in the feature space. This is the best scenario in regression; we cannot achieve a lower cost on average for this squared cost.

The next situation is when $f(\mathbf{x}) \neq \mathbb{E}[Y|\mathbf{x}]$. Here, we will proceed by decomposing the squared error as

$$\begin{aligned}(f(\mathbf{x}) - y)^2 &= (f(\mathbf{x}) - \mathbb{E}[Y|\mathbf{x}] + \mathbb{E}[Y|\mathbf{x}] - y)^2 \\ &= (f(\mathbf{x}) - \mathbb{E}[Y|\mathbf{x}])^2 + 2\underbrace{(f(\mathbf{x}) - \mathbb{E}[Y|\mathbf{x}])(\mathbb{E}[Y|\mathbf{x}] - y)}_{g(\mathbf{x},y)} + (\mathbb{E}[Y|\mathbf{x}] - y)^2\end{aligned}$$

We now take a more detailed look at $g(\mathbf{x}, y)$ when placed under the expectation over $p(\mathbf{x}, y)$

$$\begin{aligned}\mathbb{E}[g(\boldsymbol{X}, y)] &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(\mathbf{x}) - \mathbb{E}[Y|\mathbf{x}])(\mathbb{E}[Y|\mathbf{x}] - y) p(\mathbf{x}, y) dy d\mathbf{x} \\ &= \int_{\mathcal{X}} \int_{\mathcal{Y}} f(\mathbf{x}) \mathbb{E}[Y|\mathbf{x}] p(\mathbf{x}, y) dy d\mathbf{x} - \int_{\mathcal{X}} \int_{\mathcal{Y}} f(\mathbf{x}) y p(\mathbf{x}, y) dy d\mathbf{x} \\ &\quad + \int_{\mathcal{X}} \int_{\mathcal{Y}} \mathbb{E}[Y|\mathbf{x}] y p(\mathbf{x}, y) dy d\mathbf{x} - \int_{\mathcal{X}} \int_{\mathcal{Y}} \mathbb{E}[Y|\mathbf{x}] \mathbb{E}[Y|\mathbf{x}] p(\mathbf{x}, y) dy d\mathbf{x} \\ &= \int_{\mathcal{X}} f(\mathbf{x}) \mathbb{E}[Y|\mathbf{x}] p(\mathbf{x}) d\mathbf{x} - \int_{\mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) \int_{\mathcal{Y}} y p(y|\mathbf{x}) dy d\mathbf{x} \\ &\quad \int_{\mathcal{X}} \mathbb{E}[Y|\mathbf{x}] \mathbb{E}[Y|\mathbf{x}] p(\mathbf{x}) d\mathbf{x} - \int_{\mathcal{X}} \mathbb{E}[Y|\mathbf{x}] \mathbb{E}[Y|\mathbf{x}] p(\mathbf{x}) d\mathbf{x} \\ &= 0.\end{aligned}$$

Therefore, we can now express the expected cost as

$$\begin{aligned}\mathbb{E}[C] &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(\mathbf{x}) - y)^2 p(\mathbf{x}, y) dy d\mathbf{x} \\ &= \int_{\mathcal{X}} (f(\mathbf{x}) - \mathbb{E}[Y|\mathbf{x}])^2 p(\mathbf{x}) d\mathbf{x} + \int_{\mathcal{X}} \int_{\mathcal{Y}} (\mathbb{E}[Y|\mathbf{x}] - y)^2 p(\mathbf{x}, y) dy d\mathbf{x},\end{aligned}$$

where the first term is the "distance" between the trained model $f(\mathbf{x})$ and the optimal model $\mathbb{E}[Y|\mathbf{x}]$ and the second term is the "distance" between the optimal model $\mathbb{E}[Y|\mathbf{x}]$ and the correct target value y. These terms are also often called the *reducible* and *irreducible* errors. If we extend the class of functions f to predict $\mathbb{E}[Y|\mathbf{x}]$, we can reduce the first expected error. However, the second distance is an inherent error, where for any inputs $\mathbf{x}$, there is a distribution over possible values that we can observe. This relates to the problem of partial observability, where there is always some stochasticity due to a lack of information.

This irreducible distance could potentially be further reduced by providing more feature information (i.e., extending the information in $\mathbf{x}$). However, for a given dataset, with the given features, this error is irreducible.

To sum up, we argued here that optimal classification and regression models critically depend on knowing or accurately learning the posterior distribution $p(y|\mathbf{x})$. This task can be solved in different ways, but a straightforward approach is to assume a functional form for $p(y|\mathbf{x})$, say $p(y|\mathbf{x}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a set of weights or parameters that are to be learned from the data. Alternatively, we can learn the class-conditional and prior distributions, $p(\mathbf{x}|y)$ and $p(y)$, respectively. Using

$$\begin{aligned} p(y|\mathbf{x}) &= \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x}|y)p(y)}{\sum_y p(\mathbf{x}, y)} \\ &= \frac{p(\mathbf{x}|y)p(y)}{\sum_y p(\mathbf{x}|y)p(y)} \end{aligned}$$

we can see that these two learning approaches are equivalent in theory. The choice depends on our prior knowledge and/or preferences.

Models obtained by directly estimating $p(y|\mathbf{x})$ are called *discriminative models* and models obtained by directly estimating $p(\mathbf{x}|y)$ and $p(y)$ are called *generative models*. Direct estimation of the joint distribution $p(\mathbf{x}, y)$ is relatively rare as it may be more difficult to hypothesize its parametric or non-parametric form from which the parameters are to be found. Finally, in some situations we can train a model without an explicit probabilistic approach in mind. In these cases, we typically aim to show a good performance of an algorithm in practice, say on a large number of data sets, according to a performance measure relevant to the problems at hand.

4.4 [Advanced] Bayes Optimal Models

We saw earlier that optimal prediction models reduce to the learning of the posterior distribution $p(y|\mathbf{x})$ which is then used to minimize the expected cost (risk, loss). However, in practice, the probability distribution $p(y|\mathbf{x})$ must be modeled using a particular functional form and a set of tunable coefficients. When $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$, one such example is used in logistic regression, where

$$p(1|\mathbf{x}) = \frac{1}{1 + e^{-(w_0 + \sum_{j=1}^{d} w_j x_j)}}$$

and $p(0|\mathbf{x}) = 1 - p(1|\mathbf{x})$. Here $(w_0, w_1, \ldots, w_d) \in \mathbb{R}^{d+1}$ is a set of weights that are to be inferred from a given data set $\mathcal{D}$ and $\mathbf{x} \in \mathbb{R}^d$ is an input data point. A number of other types of functional relationships can be used as well, providing a vast set of possibilities for modeling distributions.

To be more precise about these functional forms, we should adjust our notation to denote the distribution over y given $\mathbf{x}$ as

$$p(y|\mathbf{x}) = p(y|\mathbf{x}, f),$$

where f is a particular function from some function (hypothesis) space $\mathcal{F}$. We can think of $\mathcal{F}$ as a set of all functions from a specified class, say for all $(w_0, w_1, \ldots, w_d) \in \mathbb{R}^{k+1}$ in the example above, but we can also extend the functional class beyond simple parameter variation to incorporate non-linear decision surfaces. We typically select one function, given the data—say the maximum likelihood or MAP solution

We could instead consider the distribution $Y|\mathbf{x}$, over all plausible functions f. In a typical learning problem, we are given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and are asked to model $p(y|\mathbf{x})$. For this purpose, we will think of $\mathcal{D}$ as a realization of a random variable D and will assume that $\mathcal{D}$ was drawn according to the true underlying distribution $p(\mathbf{x}, y)$. Thus, our task is to express $p(y|\mathbf{x}, \mathcal{D})$. Using the sum and product rules, will rewrite our original task of estimating $p(y|\mathbf{x})$ as

$$\begin{aligned} p(y|\mathbf{x}, \mathcal{D}) &= \int_{\mathcal{F}} p(y|\mathbf{x}, f, \mathcal{D}) p(f|\mathbf{x}, \mathcal{D}) df \\ &= \int_{\mathcal{F}} p(y|\mathbf{x}, f) p(f|\mathbf{x}, \mathcal{D}) df. \end{aligned}$$

Here we used conditional independence between output Y and the data set D once a particular model f was selected based on $\mathcal{D}$; thus $p(y|\mathbf{x}, f, \mathcal{D}) = p(y|\mathbf{x}, f)$. This equation, gives us a sense that the optimal decision can be made through a mixture of distributions $p(y|\mathbf{x}, f)$, where the weights are given as posterior densities $p(f|\mathbf{x}, \mathcal{D})$. In finite hypothesis spaces $\mathcal{F}$ we have that

$$p(y|\mathbf{x}, \mathcal{D}) = \sum_{f \in \mathcal{F}} p(y|\mathbf{x}, f) p(f|\mathbf{x}, \mathcal{D}),$$

and $p(f|\mathbf{x}, \mathcal{D})$ are posterior probabilities. We may further assume that $p(f|\mathbf{x}, \mathcal{D}) = p(f|\mathcal{D})$, in which case the weights can be precomputed based on the given data set $\mathcal{D}$. This leads to more efficient calculations of the posterior probabilities.

In classification, we can rewrite our original MAP classifier as

$$f_{\text{MAP}}(\mathbf{x}, \mathcal{D}) = \arg\max_{y \in \mathcal{Y}} \left\{ p(y|\mathbf{x}, \mathcal{D}) \right\},$$

which readily leads to the following formulation

$$\begin{aligned} f_{\text{MAP}}(\mathbf{x}, \mathcal{D}) &= \arg\max_{y \in \mathcal{Y}} \left\{ \int_{\mathcal{F}} p(y|\mathbf{x}, f, \mathcal{D}) p(f|\mathcal{D}) df \right\} \\ &= \arg\max_{y \in \mathcal{Y}} \left\{ \int_{\mathcal{F}} p(y|\mathbf{x}, f) p(\mathcal{D}|f) p(f) df \right\}. \end{aligned}$$

It can be shown that no classifier can outperform the *Bayes optimal classifier*. Interestingly, the Bayes optimal model also hints that a better prediction performance can be achieved by combining multiple models and averaging their outputs. This provides theoretical support for ensemble learning and methods such as bagging and boosting.

One problem in Bayes optimal classification is efficient calculation of $f_{\text{MAP}}(\mathbf{x}, \mathcal{D})$, given that the function (hypothesis) space $\mathcal{F}$ is generally uncountable. One approach to this is sampling of functions from $\mathcal{F}$ according to $p(f)$ and then calculating $p(f|\mathcal{D})$ or $p(\mathcal{D}|f)$. This can be computed until $p(y|\mathbf{x}, \mathcal{D})$ converges.

Chapter 5

Linear Regression

Given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ the objective is to learn the relationship between features and the target. We usually start by hypothesizing the functional form of this relationship. For example,

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

where $\mathbf{w} = (w_0, w_1, w_2)$ is a set of parameters that need to be determined (learned) and $\mathbf{x} = (x_1, x_2)$. Alternatively, we may hypothesize that $f(\mathbf{x}) = \alpha + \beta x_1 x_2$, where $\boldsymbol{\theta} = (\alpha, \beta)$ is another set of parameters to be learned. In the former case, the target function is modeled as a *linear combination* of features and parameters, i.e.

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

where we extended $\mathbf{x}$ to $(x_0 = 1, x_1, x_2, \ldots, x_d)$. Finding the best parameters $\mathbf{w}$ is then referred to as *linear regression problem*, whereas all other types of relationship between the features and the target fall into a category of *non-linear regression*. In either situation, the regression problem can be presented as a probabilistic modeling approach that reduces to parameter estimation; i.e. to an optimization problem with the goal of maximizing or minimizing some performance criterion between target values $\{y_i\}_{i=1}^n$ and predictions $\{f(\mathbf{x}_i)\}_{i=1}^n$. We can think of a particular optimization algorithm as the *learning* or *training algorithm*.

5.1 Maximum likelihood formulation

We now consider a statistical formulation of linear regression. We shall first lay out the assumptions behind this process and subsequently formulate the problem through maximization of the conditional likelihood function. In following section, we will show how to solve the optimization and analyze the solution and its basic statistical properties.

Let us assume that the observed data set $\mathcal{D}$ is a product of a data generating process in which n data points were drawn independently and according to the same distribution $p(\mathbf{x})$. Assume also that the target variable Y has an underlying linear relationship with features $\boldsymbol{X} = (X_1, X_2, \ldots, X_d)$, modified by some error term ε that follows a zero-mean Gaussian distribution, i.e. $\varepsilon : \mathcal{N}(0, \sigma^2)$. That is, for a given input $\mathbf{x}$, the target y is a realization of a random variable Y defined as

$$Y = \sum_{j=0}^{d} \omega_j X_j + \varepsilon,$$

where $\boldsymbol{\omega} = (\omega_0, \omega_1, \ldots, \omega_d)$ is a set of unknown coefficients we seek to recover through estimation. Generally, the assumption of normality for the error term is reasonable (recall the central limit theorem!), although the independence between ε and $\boldsymbol{X}$ may not hold in practice. Using a few simple properties of expectations, we can see that Y also follows a Gaussian distribution, i.e. its conditional density is $p(y|\mathbf{x}, \boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}^\top \mathbf{x}, \sigma^2)$.

In linear regression, we seek to approximate the target as $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$, where weights $\mathbf{w}$ are to be determined. We first write the conditional likelihood function for a single pair $(\mathbf{x}, y)$ as

$$p(y|\mathbf{x}, \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y - \sum_{j=0}^{d} w_j x_j\right)^2}{2\sigma^2}\right)$$

where we use the notation $\exp(a) = e^a$, to make the exponent easier to read. Observe that the only change from the conditional density function of Y is that coefficients $\mathbf{w}$ are used instead of $\boldsymbol{\omega}$. Incorporating the entire data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we can now write the conditional likelihood function as $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ and find weights as

$$\mathbf{w}_{\text{ML}} = \arg\max_{\mathbf{w}} \{p(\mathbf{y}|\mathbf{X}, \mathbf{w})\}.$$

Since the n examples are independent and identically distributed (i.i.d.), we have

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - \sum_{j=0}^{d} w_j x_{ij}\right)^2}{2\sigma^2}\right). \end{aligned}$$

For the reasons of mathematical convenience, we will look at the logarithm (monotonic function) of the likelihood function and express the log-likelihood as

$$\ln(p(\mathbf{y}|\mathbf{X}, \mathbf{w})) = -\sum_{i=1}^{n} \log\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{d} w_j x_{ij}\right)^2.$$

Given that the first term on the right-hand hand side is independent of $\mathbf{w}$, maximizing the likelihood function corresponds exactly to minimizing the sum of squared errors

$$\begin{aligned} \text{Err}(\mathbf{w}) &= \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2 \qquad \triangleright\, f(\mathbf{x}_i) = \sum_{j=0}^{d} w_j x_{ij} \\ &= \sum_{i=1}^{n} e_i^2. \end{aligned}$$

Geometrically, this error is the square of the Euclidean distance between the vector of predictions $\hat{\mathbf{y}} = (f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n))$ and the vector of observed target values $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. A simple example illustrating the linear regression problem is shown in Figure 5.1.

To more explicitly see why the maximum likelihood solution corresponds to minimizing $\text{Err}(\mathbf{w})$, notice that maximizing the likelihood is equivalent to maximizing the log-likelihood

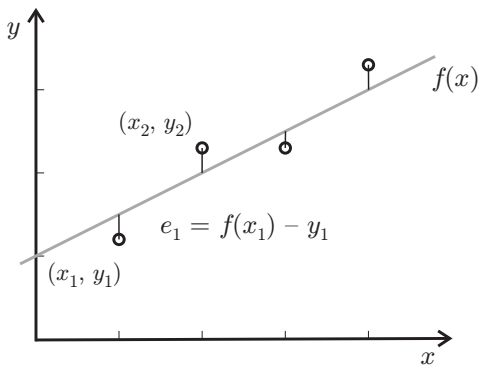


Figure 5.1: An example of a linear regression fitting on data set $\mathcal{D} = \{(1, 1.2), (2, 2.3), (3, 2.3), (4, 3.3)\}$. *The task of the optimization process is to find the best linear function* $f(x) = w_0 + w_1 x$ *so that the sum of squared errors* $e_1^2 + e_2^2 + e_3^2 + e_4^2$ *is minimized.*

(because log is monotonic) which is equivalent to minimizing the negative log-likelihood. Therefore, the maximum likelihood $\mathbf{w}_{\text{ML}}$ corresponds to

$$\begin{aligned}
\mathbf{w}_{\text{ML}} &= \underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}} - \ln(p(\mathbf{y}|\mathbf{X}, \mathbf{w})) \\
&= \underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}} \sum_{i=1}^{n} \log\left(\sqrt{2\pi\sigma^2}\right) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{d} w_j x_{ij}\right)^2 \\
&= \underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}} \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{d} w_j x_{ij}\right)^2 \\
&= \underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}}\, \text{Err}(\mathbf{w})
\end{aligned}$$

In the next sections, we will discuss how to solve this optimization and the properties of the solution.

Note that we could have simply started with some (expert-defined) error function, as was originally done for OLS and using $\text{Err}(\mathbf{w})$. However, the statistical framework provides insights into the assumptions behind OLS regression. In particular, the assumptions include that the data $\mathcal{D}$ was drawn i.i.d.; there is an underlying linear relationship between features and the target; that the noise (error term) is zero-mean Gaussian and independent of the features; and that there is an absence of noise in the collection of features.

5.2 Ordinary Least-Squares (OLS) Regression

To minimize the sum of squared errors, we shall first re-write $\text{Err}(\mathbf{w})$ as

$$\begin{aligned}\text{Err}(\mathbf{w}) &= \sum_{i=1}^{n} \left(f(\mathbf{x}_i) - y_i\right)^2 \\ &= \sum_{i=1}^{n} \left(\sum_{j=0}^{d} w_j x_{ij} - y_i\right)^2,\end{aligned}$$

where, again, we expanded each data point $\mathbf{x}_i$ by $x_{i0} = 1$ to simplify the expression.

We now calculate the gradient $\nabla\text{Err}(\mathbf{w})$. Finding weights for which $\nabla\text{Err}(\mathbf{w}) = \mathbf{0}$ will result in a stationary point. To ensure that this stationary point is a global minimum, we need a bit more information. We can look at the second derivative; this requires understanding of Hessian, so we include this later in the notes in Example 16. But, fortunately, it is even simpler here, since we know that this objective is convex in $\mathbf{w}$; therefore, any stationary point will be a global minimum.

Now, we set the partial derivatives to 0 and solve the equations for each weight w_j

$$\begin{aligned}\frac{\partial \text{Err}}{\partial w_0} &= 2\sum_{i=1}^{n} \left(\sum_{j=0}^{d} w_j x_{ij} - y_i\right) x_{i0} = 0 \\ \frac{\partial \text{Err}}{\partial w_1} &= 2\sum_{i=1}^{n} \left(\sum_{j=0}^{d} w_j x_{ij} - y_i\right) x_{i1} = 0 \\ &\vdots \\ \frac{\partial \text{Err}}{\partial w_d} &= 2\sum_{i=1}^{n} \left(\sum_{j=0}^{d} w_j x_{ij} - y_i\right) x_{ik} = 0\end{aligned}$$

This results in a system of $d+1$ linear equations with $d+1$ unknowns that can be routinely solved (e.g. by using Gaussian elimination).

While this formulation is useful, it does not allow us to obtain a closed-form solution for $\mathbf{w}$ or discuss the existence or multiplicity of solutions. To address the first point we will exercise some matrix calculus, while the remaining points will be discussed later. We will first write the sum of square errors using the matrix notation as

$$\begin{aligned}\text{Err}(\mathbf{w}) &= (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2,\end{aligned}$$

where $\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}^\top \mathbf{v}} = \sqrt{v_1^2 + v_2^2 + \ldots v_n^2}$ is the length of vector $\mathbf{v}$; it is also called the ℓ_2 norm. We can now formalize the *ordinary least-squares (OLS) linear regression problem* as

$$\mathbf{w}_{\text{ML}} = \arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2.$$

We proceed by finding $\nabla \text{Err}(\mathbf{w})$. The gradient function $\nabla \text{Err}(\mathbf{w})$ is a derivative of a scalar with respect to a vector. However, the intermediate steps of calculating the gradient require derivatives of vectors with respect to vectors (some of the rules of such derivatives are shown in Table B.1). Application of the rules from Table B.1 results in

$$\nabla \text{Err}(\mathbf{w}) = 2\mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{X}^\top \mathbf{y}$$

and, therefore, from $\nabla \text{Err}(\mathbf{w}) = \mathbf{0}$ we find that

$$\mathbf{w}_{\text{ML}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}. \tag{5.1}$$

We can now express the predicted target values as

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\mathbf{w}_{\text{ML}} \\ &= \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}.\end{aligned}$$

The matrix $\mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$ is called the *projection matrix*; see Section C.1 to understand how it projects $\mathbf{y}$ to the column space of $\mathbf{X}$.

Example 13: Consider again data set $\mathcal{D} = \{(1, 1.2), (2, 2.3), (3, 2.3), (4, 3.3)\}$ from Figure 5.1. We want to find the optimal coefficients of the least-squares fit for $f(x) = w_0 + w_1 x$ and then calculate the sum of squared errors on $\mathcal{D}$ after the fit.

The OLS fitting can now be performed using

$$\mathbf{x} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \ \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} 1.2 \\ 2.3 \\ 2.3 \\ 3.3 \end{bmatrix},$$

where a column of ones was added to $\mathbf{x}$ to allow for a non-zero intercept ($y = w_0$ when $x = 0$). Substituting $\mathbf{x}$ and $\mathbf{y}$ into Eq. (5.1) results in $\mathbf{w} = (0.7, 0.63)$ and the sum of square errors is $\text{Err}(\mathbf{w}) = 0.223$. □

As seen in the example above, it is a standard practice to add a column of ones to the data matrix $\mathbf{x}$ in order to ensure that the fitted line, or generally a hyperplane, does not have to pass through the origin of the coordinate system. This effect, however, can be achieved in other ways. Consider the first component of the gradient vector

$$\frac{\partial \text{Err}}{\partial w_0} = 2\sum_{i=1}^{n}\left(\sum_{j=0}^{d} w_j x_{ij} - y_i\right) x_{i0} = 0$$

where, because $x_{i0} = 1$ by definition, we obtain that

$$0 = \sum_{i=1}^{n}\left(\sum_{j=0}^{d} w_j x_{ij} - y_i\right) = \sum_{i=1}^{n}\left(w_0 + \sum_{j=1}^{d} w_j x_{ij} - y_i\right)$$

giving

$$\sum_{i=1}^{n} w_0 = \sum_{i=1}^{n} y_i - \sum_{j=1}^{d} w_j \sum_{i=1}^{n} x_{ij}.$$

When all features (columns of $\mathbf{x}$) are normalized to have zero mean, i.e. when $\sum_{i=1}^{n} x_{ij} = 0$ for any column j, it follows that

$$w_0 = \frac{1}{n} \sum_{i=1}^{n} y_i.$$

We see now that if the target variable is normalized to the zero mean as well, it follows that $w_0 = 0$ and that the column of ones is not needed.

5.2.1 Weighted error function

In some applications it is useful to consider minimizing the weighted error function

$$\mathrm{Err}(\mathbf{w}) = \sum_{i=1}^{n} c_i \left(\sum_{j=0}^{d} w_j x_{ij} - y_i \right)^2,$$

where $c_i > 0$ is a cost for data point i. Expressing this in a matrix form, the goal is to minimize $(\mathbf{X}\mathbf{w} - \mathbf{y})^\top \mathbf{C} (\mathbf{X}\mathbf{w} - \mathbf{y})$, where $\mathbf{C} = \mathrm{diag}(c_1, c_2, \ldots, c_n)$. Using a similar approach as above, it can be shown that the weighted least-squares solution $\mathbf{w}_{\mathrm{C}}$ can be expressed as

$$\mathbf{w}_{\mathrm{C}} = \left(\mathbf{X}^\top \mathbf{C} \mathbf{X} \right)^{-1} \mathbf{X}^\top \mathbf{C} \mathbf{y}.$$

In addition, it can be derived that

$$\mathbf{w}_{\mathrm{C}} = \mathbf{w}_{\mathrm{ML}} + \left(\mathbf{X}^\top \mathbf{C} \mathbf{X} \right)^{-1} \mathbf{X}^\top (\mathbf{I} - \mathbf{C}) (\mathbf{X}\mathbf{w}_{\mathrm{ML}} - \mathbf{y}),$$

where $\mathbf{w}_{\mathrm{ML}}$ is provided by Eq. (5.1). We can see that the solutions are identical when $\mathbf{C} = \mathbf{I}$, but also when $\mathbf{X}\mathbf{w}_{\mathrm{ML}} = \mathbf{y}$.

5.2.2 Predicting multiple outputs simultaneously

The extension to multiple outputs is straightforward, where now the target is an m-dimensional vector, $\mathbf{y} \in \mathbb{R}^m$, rather than a scalar, giving target matrix $\mathbf{Y} \in \mathbb{R}^{n \times m}$. Correspondingly, the weights $\mathbf{W} \in \mathbb{R}^{d \times m}$ to give $\mathbf{x}\mathbf{W} \in \mathbb{R}^m$, with error

$$\begin{aligned} \mathrm{Err}(\mathbf{W}) &= \|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_F^2 = \sum_{i=1}^{n} \|\mathbf{X}_{i,:}\mathbf{W} - \mathbf{Y}_{i,:}\|_2^2 && \triangleright \text{ Frobenius norm} \\ &= \mathrm{trace}\left((\mathbf{X}\mathbf{W} - \mathbf{Y})^\top (\mathbf{X}\mathbf{W} - \mathbf{Y}) \right) \end{aligned}$$

and solution

$$\mathbf{W}_{\mathrm{ML}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}.$$

Exercise: Derive this solution, by taking partial derivatives or, preferably, by using gradient rules for matrix variables. A good resource for matrix gradients is the matrix cookbook [16].

To gain further insight into the ordinary linear regression solution, see an algebraic perspective in the appendix C.1. It provides more insights into uniqueness of the solution, and the space of possible solutions, and connects the linear regression optimization to solving systems.

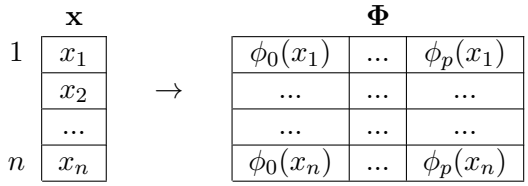


Figure 5.2: Transformation of an $n \times 1$ data matrix $\mathbf{x}$ into an $n \times (p+1)$ matrix $\mathbf{\Phi}$ using a set of basis functions ϕ_j, $j = 0, 1, \ldots, p$.

5.3 Linear regression for non-linear problems

At first, it might seem that the applicability of linear regression to real-life problems is greatly limited. After all, it is not clear whether it is realistic (most of the time) to assume that the target variable is a linear combination of features. Fortunately, the applicability of linear regression is broader, because we can use it to obtain non-linear functions. The main idea is to apply a non-linear transformation to the data matrix $\mathbf{X}$ prior to the fitting step, which then enables a non-linear fit. Obtaining such a useful feature representation is a central problem in machine learning; we will discuss this in detail in Chapter 9. Here, we will first examine a simpler expanded representation that enables non-linear learning: polynomial curve fitting.

5.3.1 Polynomial curve fitting

We start with one-dimensional data. In OLS regression, we would look for the fit in the following form

$$f(x) = w_0 + w_1 x,$$

where x is the data point and $\mathbf{w} = (w_0, w_1)$ is the weight vector. To achieve a polynomial fit of degree p, we will modify the previous expression into

$$f(x) = \sum_{j=0}^{p} w_j x^j,$$

where p is the degree of the polynomial. We will rewrite this expression using a set of basis functions as

$$\begin{aligned} f(x) &= \sum_{j=0}^{p} w_j \phi_j(x) \\ &= \mathbf{w}^\top \boldsymbol{\phi}, \end{aligned}$$

where $\phi_j(x) = x^j$ and $\boldsymbol{\phi} = (\phi_0(x), \phi_1(x), \ldots, \phi_p(x))$. Applying this transformation to every data point in $\mathbf{x}$ results in a new data matrix $\mathbf{\Phi}$, as shown in Figure 5.2.

Following the discussion from Section 5.2, the optimal set of weights is calculated as

$$\mathbf{w}_{\text{ML}} = \left(\mathbf{\Phi}^\top \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^\top \mathbf{y}.$$

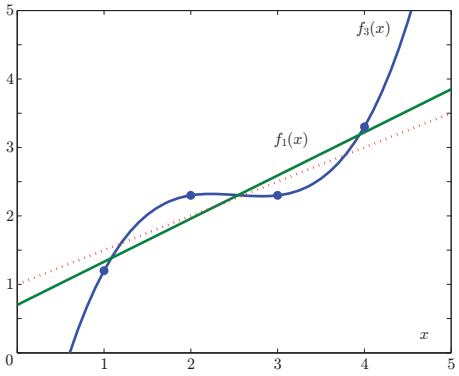


Figure 5.3: Example of a linear vs. polynomial fit on a data set shown in Figure 5.1. The linear fit, $f_1(x)$, is shown as a solid green line, whereas the cubic polynomial fit, $f_3(x)$, is shown as a solid blue line. The dotted red line indicates the target linear concept.

Example 14: In Figure 5.1 we presented an example of a data set with four data points. What we did not mention was that, given a set $\{x_1, x_2, x_3, x_4\}$, the targets were generated by using function $1 + \frac{x}{2}$ and then adding a measurement error $\mathbf{e} = (-0.3, 0.3, -0.2, 0.3)$. It turned out that the optimal coefficients $\mathbf{w}_{\text{ML}} = (0.7, 0.63)$ were close to the true coefficients $\boldsymbol{\omega} = (1, 0.5)$, even though the error terms were relatively significant. We will now attempt to estimate the coefficients of a polynomial fit with degrees $p = 2$ and $p = 3$. We will also calculate the sum of squared errors on $\mathcal{D}$ after the fit as well as on a large discrete set of values $x \in \{0, 0.1, 0.2, \ldots, 10\}$ where the target values will be generated using the true function $1 + \frac{x}{2}$.

Using a polynomial fit with degrees $p = 2$ and $p = 3$ results in $\mathbf{w}_2 = (0.575, 0.755, -0.025)$ and $\mathbf{w}_3 = (-3.1, 6.6, -2.65, 0.35)$, respectively. The sum of squared errors on $\mathcal{D}$ equals $\text{Err}(\mathbf{w}_2) = 0.221$ and $\text{Err}(\mathbf{w}_3) \approx 0$. Thus, the best fit is achieved with the cubic polynomial. However, the sum of squared errors on the outside data set reveal a poor generalization ability of the cubic model because we obtain $\text{Err}(\mathbf{w}) = 26.9$, $\text{Err}(\mathbf{w}_2) = 3.9$, and $\text{Err}(\mathbf{w}_3) = 22018.5$. This effect is called *overfitting*. Broadly speaking, overfitting is indicated by a significant difference in fit between the data set on which the model was trained and the outside data set on which the model is expected to be applied (Figure 5.3). In this case, the overfitting occurred because the complexity of the model was increased considerably, whereas the size of the data set remained small.

One signature of overfitting is an increase in the magnitude of the coefficients. For example, while the absolute values of all coefficients in $\mathbf{w}$ and $\mathbf{w}_2$ were less than one, the values of the coefficients in $\mathbf{w}_3$ became significantly larger with alternating signs (suggesting overcompensation). We will discuss *regularization* in Section 5.4.2 as an approach to prevent this effect. □

Polynomial curve fitting is only one way of non-linear fitting because the choice of basis functions need not be limited to powers of x. Among others, non-linear basis functions that

are commonly used are the sigmoid function

$$\phi_j(x) = \frac{1}{1 + e^{-\frac{x-\mu_j}{s_j}}}$$

or a Gaussian-style exponential function

$$\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}},$$

where μ_j, s_j, and σ_j are constants to be determined. However, this approach works only for a one-dimensional input $\mathbf{x}$. For higher dimensions, this approach can be generalized using *radial basis functions*; see Section 9.1 for more details.

5.4 Stability and the bias-variance trade-off

The OLS solution can be unstable. In this section, we show why this is the case, and discuss how regularization can be used to mitigate this problem. We will then discuss a foundational concept in machine learning: the bias-variance trade-off.

5.4.1 Sensitivity of the OLS solution

The OLS solution is unstable if $\mathbf{X}^\top\mathbf{X}$ is not invertible. This can occur for two main reasons: linearly dependent features and small datasets. Data sets often include large numbers of features, which are sometimes identical, similar, or nearly linearly dependent. If the dataset is small, it is feasible that some features are the same across samples, again resulting in low-rank $\mathbf{X}$. When $\mathbf{X}^\top\mathbf{X}$ is not invertible—or ill-conditioned—the OLS solution is highly sensitive to small perturbations in $\mathbf{y}$ and $\mathbf{X}$.

To see why, we will look at the *singular value decomposition* of $\mathbf{X}$. As with the previous linear algebra constructs, it allows us to easily examine properties of $\mathbf{X}$. Let's consider the common case, where $n > d$: the number of samples is greater than the input dimension. The singular value decomposition of $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ for orthonormal matrices[1] $\mathbf{U} \in \mathbb{R}^{n\times n}$, $\mathbf{V} \in \mathbb{R}^{d\times d}$ and non-negative (rectangular) diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{n\times d}$. The diagonal entries in $\mathbf{\Sigma}$ are the singular values, which we typically order in descending order $\sigma_1, \sigma_2, \ldots, \sigma_d$, giving

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & \ldots & 0 \\ & & \vdots & & \\ 0 & 0 & \ldots & 0 & \sigma_d \\ 0 & 0 & \ldots & 0 & 0 \\ & & \vdots & (n-d) \text{ rows} & \text{of zeros} \\ 0 & 0 & \ldots & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma}_d \\ \mathbf{0} \end{bmatrix} \qquad \text{where } \mathbf{\Sigma}_d = \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ & & \vdots & \\ 0 & 0 & \ldots & \sigma_d \end{bmatrix}.$$

Any matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ can be decomposed into its singular value decomposition, because any linear transformation can be decomposed into a rotation (multiplication by $\mathbf{V}^\top$), followed by a scaling (multiplication by $\mathbf{\Sigma}$), followed again by a rotation (multiplication by $\mathbf{U}$).

[1] An orthonormal matrix $\mathbf{U}$ is square matrix that satisfies $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ and $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$

This decomposition simplifies analysis of the properties of a matrix. For example, the number of non-zero singular values constitutes the rank of $\mathbf{X}$. To see why, assume $\sigma_d = 0$, and $\sigma_{d-1} > 0$, meaning $\mathbf{X}$ has rank $d-1$. Take any vector $\mathbf{w} \in \mathbb{R}^d$, and consider $\mathbf{X}\mathbf{w}$. We can write this product as $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{w} = \mathbf{U}\mathbf{\Sigma}\tilde{\mathbf{w}}$ for $\tilde{\mathbf{w}} = \mathbf{V}^\top\mathbf{w}$. The product $\mathbf{\Sigma}\tilde{\mathbf{w}}$ sets the last dimension of $\tilde{\mathbf{w}}$ to zero, effectively removing that dimension and so projecting $\tilde{\mathbf{w}}$ into a lower-dimensional $(d-1)$ space. Then it rotates that projected vector afterwards, using $\mathbf{U}$, but cannot undo that projection into a lower-dimensional space. Therefore, $\mathbf{X}\mathbf{w}$ can only product $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ that lie in a $d-1$-dimensional plane, rotated in $\mathbb{R}^{d-1}$. This decomposition, then, can help us understand the space of possible predictions for linear regression $\mathbf{X}\mathbf{w}$.

Now we can discuss the least-squares solution, in terms of the singular value decomposition of $\mathbf{X}$. Notice that

$$\mathbf{X}^\top\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^\top\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}_d^2\mathbf{V}^\top$$

because $\mathbf{U}$ is orthonormal and so $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ the identity matrix ($\mathbf{I}$ is a diagonal matrix with ones on the diagonal). The inverse of $\mathbf{X}^\top\mathbf{X}$ exists if $\mathbf{X}$ is full rank, i.e., $\mathbf{\Sigma}_d$ has no zeros on the diagonal, because $(\mathbf{X}^\top\mathbf{X})^{-1} = \mathbf{V}\mathbf{\Sigma}_d^{-2}\mathbf{V}^\top$. The resulting solution for $\mathbf{w}$ looks like[2]

$$\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{y} = \sum_{j=1}^{d}\frac{\mathbf{u}_j^\top\mathbf{y}}{\sigma_j}\mathbf{v}_j \tag{5.2}$$

where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n] \in \mathbb{R}^{n\times n}$ is the orthonormal matrix composed of the left singular vectors, $\mathbf{U}_d = [\mathbf{u}_1, \ldots, \mathbf{u}_d] \in \mathbb{R}^{n\times d}$ is the first d left singular vectors, and $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_d] \in \mathbb{R}^{d\times d}$ is the orthonormal matrix composed of the right singular vectors.

The solution in Equation (5.3) makes it clear why the linear regression solution can be sensitive to perturbations. For small singular values, σ_j^{-1} is large and amplifies any changes in $\mathbf{y}$. For example, for slightly different noise component ϵ_i for the ith sample, the solution vector $\mathbf{w}$ could be very different. A common strategy to deal with this instability is to drop or truncate small singular values. This is a form of regularization, which we discuss in the next section.

Remark: In the general case, where $\mathbf{X}$ is not full rank, we can still obtain a least-squares solution to $\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y}$. Now, there are potentially infinitely many solutions. The common choice is to select the minimum variance solution, which corresponds to dropping the components (singular vectors) for the zero singular values:

$$\mathbf{w} = \sum_{j=1}^{\text{rank of } \mathbf{X}}\frac{\mathbf{u}_j^\top\mathbf{y}}{\sigma_j}\mathbf{v}_j. \tag{5.3}$$

Example 15: [Nearly linear dependent] Let's look at a simple example of why $\mathbf{X} \in \mathbb{R}^{n\times d}$ might have small singular values. First, assume $d = 2$ and $x_2 = x_1$, i.e., that the second features is a copy of the first and simply redundant. Then $\mathbf{X} = \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}^\top$ is the thin SVD of $\mathbf{X}$, where $\mathbf{U}_2$ only has the first two columns of the full SVD. We can write this thin SVD because $\mathbf{X} = \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where the zero singular values zero out the remaining columns of $\mathbf{U}$.

[2]The last step in the below equation, writing the matrix product as a sum, is not immediately obvious. As an exercise, see if you can derive this last equality.

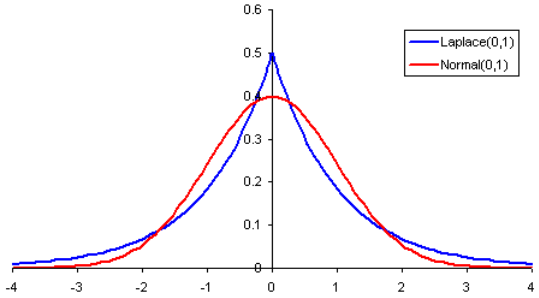


Figure 5.4: A comparison between Gaussian and Laplace priors. Both prefers values to be near zero, but the Laplace prior more strongly prefers the values to equal zero.

The SVD of just the first column $\mathbf{x}_1 \in \mathbb{R}^{n\times 1}$ is straightforward: $\mathbf{x}_1 = \mathbf{u}_1\sigma_1 v_1$, where $\mathbf{u}_1 = \mathbf{x}_1/\|\mathbf{x}_1\|$, $\sigma_1 = \|\mathbf{x}_1\|$ and $v_1 = 1$. The SVD of $\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2]$ is therefore, for any n-dimensional unit vector $\mathbf{u}_2$ that is orthogonal to $\mathbf{u}_1$, and right singular vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2$,

$$\mathbf{X} = [\mathbf{u}_1 \quad \mathbf{u}_2]\mathbf{\Sigma}[\mathbf{v}_1 \quad \mathbf{v}_2]^\top = [\mathbf{u}_1 \quad \mathbf{u}_2]\begin{bmatrix} 2\sigma_1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix} = \mathbf{u}_1\sigma_1[1.0 \quad 1.0]$$

where we extended v_1 to two-dimensions (since $d = 2$), and defined $\mathbf{v}_2$ to be orthogonal to that vector, and had to rescale σ_1 to maintain unit singular vectors. So because $\mathbf{x}_2$ is dependent on $\mathbf{x}_1$, the rank does not increase when we add it as a column and the singular value $\sigma_2 = 0$.

If instead $\mathbf{x}_2 = \mathbf{x}_1 + \epsilon$ for a small noise vector $\epsilon \in \mathbb{R}^n$, then instead we would find that σ_2 would no longer be zero, but would be very close to zero, because $\mathbf{u}_1$ and the first singular value σ_1 would largely be able to recreate $\mathbf{x}_2$. □

5.4.2 Regularization

So far, we have discussed linear regression in terms of maximum likelihood. But, as before, we can also propose a MAP objective. Instead of specifying no prior over $\mathbf{w}$, we can select a prior to help *regularize* overfitting to the observed data. We will discuss two common priors (regularizers): the Gaussian prior (ℓ_2 norm) and the Laplace prior (ℓ_1 norm), shown in Figure 5.4.

Taking the log of the zero-mean Gaussian prior, $\mathcal{N}(\mathbf{0}, \lambda^{-1}\mathbf{I})$, we get

$$-\ln p(\mathbf{w}) = \ln(2\pi|\lambda^{-1}\mathbf{I}|) + \frac{\mathbf{w}^\top\mathbf{w}}{2\lambda^{-1}} = \ln(2\pi) - d\ln(\lambda) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}.$$

because $|\lambda^{-1}\mathbf{I}| = \lambda^{-d}$, where $|\mathbf{A}|$ is the determinant of the matrix $\mathbf{A}$. As before, we can drop the first constant which does not affect the selection of $\mathbf{w}$.

Now we can combine the negative log-likelihood and the negative log prior. Then ignoring constants, we can add up the negative log-likelihood and negative log to the prior

to get

$$\begin{aligned}\operatorname*{argmin}_{\mathbf{w}} -\ln(p(\mathbf{y}|\mathbf{X},\mathbf{w})) - \ln p(\mathbf{w}) &= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{d} w_j x_{ij}\right)^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w} \\ &= \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{d} w_j x_{ij}\right)^2 + \frac{\lambda\sigma^2}{2}\mathbf{w}^\top\mathbf{w}.\end{aligned}$$

Therefore if we assume that the weights have a zero-mean Gaussian prior $\mathcal{N}(\mathbf{0}, \lambda^{-1}\sigma^2\mathbf{I})$, then we get the following ridge regression problem:

$$c(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}^\top\mathbf{w} \qquad \triangleright\, \|\mathbf{w}\|_2^2 = \mathbf{w}^\top\mathbf{w}$$

where λ is a user-selected parameter that is called the *regularization parameter*. The idea is to penalize weight coefficients that are too large; the larger the λ, the more large weights are penalized. Correspondingly, larger λ corresponds to a smaller covariance in the prior, pushing the weights to stay near zero. The MAP estimate, therefore, has to balance between this prior on the weights, and fitting the observed data.

If we solve this equation in a similar manner as before, we obtain

$$\mathbf{w}_{\text{MAP}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}.$$

This has the nice effect of shifting the squared singular values in Σ_d^2 by λ, removing stability issues with dividing by small singular values, as long as λ is itself large enough.

If we choose a Laplace distribution, we get an ℓ_1 penalized objective

$$c(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\|\mathbf{w}\|_1$$

which is often called the Lasso. This objective can be obtained similarly to the ℓ_2 regularized objective, but instead using a Laplace distribution with parameter λ for the prior. As with the ℓ_2 regularizer for ridge regression, this regularizer penalizes large values in $\mathbf{w}$. However, it also produces more sparse solutions, where entries in $\mathbf{w}$ are zero. This preference can be seen in Figure 5.4, where the Laplace distribution is more concentrated around zero. In practice, however, this preference is even stronger than implied by the distribution, due to how the spherical least-squares loss and the ℓ_1 regularizer interact.

Forcing entries in $\mathbf{w}$ to zero has the effect of feature selection, because zeroing entries in $\mathbf{w}$ is equivalent to removing the corresponding feature. Consider the dot product each time a prediction is made,

$$\mathbf{x}^\top\mathbf{w} = \sum_{j=0}^{d} x_j w_j = \sum_{j:w_j\neq 0} x_j w_j.$$

This is equivalent to simply dropping entries in $\mathbf{x}$ and $\mathbf{w}$ where $w_j = 0$.

For the Lasso, we no longer have a closed-form solution. We do not have a closed form solution, because we cannot solve for $\mathbf{w}$ in closed-form that provides a stationary point. Instead, we use gradient descent to compute a solution to $\mathbf{w}$. The ℓ_1 regularizer, however, is non-differentiable at 0. Understanding how to optimize this objective requires a bit more optimization background, so we provide this algorithm in the next chapter, in Algorithm 4.

5.4.3 Expectation and variance for the regularized solution

A natural question to ask is how this regularization parameter can be selected, and the impact on the final solution vector. The selection of this regularization parameter leads to a bias-variance trade-off. To understand this trade-off, we need to understand what it means for the solution to be biased, and how to characterize the variance of the solution, across possible datasets.

Let us begin with understanding the bias and variance of the non-regularized solution, presuming that the distributional assumptions behind linear regression are true. This means that there exists a true parameter $\boldsymbol{\omega}$ such that for each of the data points $Y_i = \sum_{j=0}^{d} \omega_j X_{ij} + \varepsilon_i$, where the ε_j are i.i.d. random variables drawn according to $\mathcal{N}(0, \sigma^2)$. We can characterize the solution vector (estimator) $\mathbf{w}_{\text{ML}}$ as a random variable, where the randomness is across possible datasets that could have been observed. In this sense, we are considering the dataset $\mathcal{D}$ to be a random variable, and the solution $\mathbf{w}_{\text{ML}}(\mathcal{D})$ from that dataset as a function of this random variable.

Let us now look at the expected value (with respect to training data set $\mathcal{D}$) for the weight vector $\mathbf{w}_{\text{ML}}$, with $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$:

$$\begin{aligned}
\mathbb{E}[\mathbf{w}_{\text{ML}}(\mathcal{D})] &= \mathbb{E}\left[\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\left(\mathbf{X}\boldsymbol{\omega} + \boldsymbol{\varepsilon}\right)\right] \\
&= \mathbb{E}\left[\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}(\mathbf{X}^\top\mathbf{X})\boldsymbol{\omega}\right] + \mathbb{E}\left[\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\boldsymbol{\varepsilon}\right] \\
&= \mathbb{E}\left[\boldsymbol{\omega}\right] + \mathbb{E}\left[\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\right]\mathbb{E}\left[\boldsymbol{\varepsilon}\right] \\
&= \boldsymbol{\omega},
\end{aligned}$$

where the third equality follows from the fact that the noise terms $\boldsymbol{\varepsilon}$ are independent of the features and the last equality because ω is a constant vector (non-random) and $\mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}$. An estimator whose expected value is the true value of the parameter is called an *unbiased estimator*. The covariance matrix for the optimal set of parameters can be expressed as

$$\begin{aligned}
\text{Cov}[\mathbf{w}_{\text{ML}}(\mathcal{D})] &= \mathbb{E}\left[\left(\mathbf{w}_{\text{ML}}(\mathcal{D}) - \boldsymbol{\omega}\right)\left(\mathbf{w}_{\text{ML}}(\mathcal{D}) - \boldsymbol{\omega}\right)^\top\right] \\
&= \mathbb{E}\left[\mathbf{w}_{\text{ML}}(\mathcal{D})\mathbf{w}_{\text{ML}}(\mathcal{D})^\top\right] - \boldsymbol{\omega}\boldsymbol{\omega}^\top
\end{aligned}$$

Taking[3] $\mathbf{X}^\dagger = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top$, we have $\mathbf{w}_{\text{ML}}(\mathcal{D}) = \boldsymbol{\omega} + \mathbf{X}^\dagger\boldsymbol{\varepsilon}$, so

$$\begin{aligned}
\text{Cov}[\mathbf{w}_{\text{ML}}(\mathcal{D})] &= \mathbb{E}\left[\left(\boldsymbol{\omega} + \mathbf{X}^\dagger\boldsymbol{\varepsilon}\right)\left(\boldsymbol{\omega} + \mathbf{X}^\dagger\boldsymbol{\varepsilon}\right)^\top\right] - \boldsymbol{\omega}\boldsymbol{\omega}^\top \\
&= \boldsymbol{\omega}\boldsymbol{\omega}^\top + \mathbb{E}\left[\mathbf{X}^\dagger\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\mathbf{X}^{\dagger\top}\right] - \boldsymbol{\omega}\boldsymbol{\omega}^\top
\end{aligned}$$

because $\mathbb{E}\left[\mathbf{X}^\dagger\boldsymbol{\varepsilon}\boldsymbol{\omega}^\top\right] = \mathbb{E}\left[\mathbf{X}^\dagger\right]\mathbb{E}\left[\boldsymbol{\varepsilon}\right]\boldsymbol{\omega}^\top = \mathbf{0}$. Now because the noise terms are independent of the inputs, i.e., $\mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top|\mathbf{X}\right] = \mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top\right] = \sigma^2\mathbf{I}$, we can use the law of total probability (also

[3] This matrix is called the pseudo-inverse of $\mathbf{X}$. The idea of a pseudo-inverse generalizes the concept of inverses to non-invertible matrices, including rectangular matrices. It is a useful concept, but not one we will need to use again and so is not explained in-depth here.

called the tower rule), to get

$$\begin{aligned}\mathbb{E}\left[\mathbf{X}^\dagger \boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top \mathbf{X}^{\dagger\top}\right] &= \mathbb{E}\left[\mathbb{E}\left[\mathbf{X}^\dagger \boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top \mathbf{X}^{\dagger\top} | \mathbf{X}\right]\right] \\ &= \mathbb{E}\left[\mathbf{X}^\dagger \mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top | \mathbf{X}\right] \mathbf{X}^{\dagger\top}\right] \\ &= \sigma^2 \mathbb{E}\left[\mathbf{X}^\dagger \mathbf{X}^{\dagger\top}\right].\end{aligned}$$

Thus, we have

$$\mathrm{Cov}[\mathbf{w}_{\mathrm{ML}}(\mathcal{D})] = \sigma^2 \mathbb{E}\left[(\mathbf{X}^\top \mathbf{X})^{-1}\right].$$

It can be shown that estimator $\mathbf{w}_{\mathrm{ML}}(\mathcal{D}) = \mathbf{X}^\dagger \mathbf{y}$ is the one with the smallest variance among all unbiased estimators (Gauss-Markov theorem).

Unfortunately, however, as discussed above, the matrix $\mathbf{X}^\top \mathbf{X} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{V}^\top$ can be poorly conditioned, with some zero or near-zero singular values. Consequently, this covariance matrix can be poorly conditioned, with high magnitude co-variance values. This implies that, across datasets, the solution $\mathbf{w}_{\mathrm{ML}}(\mathcal{D})$ can vary widely. This type of behavior is suggestive of overfitting, and is not desirable. If our solution could be very different across several different random subsets of data, we cannot be confident in any one of these solutions.

The regularized solution, on the other hand, is much less likely to have high covariance, but will no longer be unbiased. Let $\mathbf{w}_{\mathrm{MAP}}(\mathcal{D})$ be the MAP estimate for the ℓ_2 regularized problem with some $\lambda > 0$. Using a similar analysis to above, the expected value of $\mathbf{w}_{\mathrm{MAP}}(\mathcal{D})$ is

$$\begin{aligned}\mathbb{E}[\mathbf{w}_{\mathrm{MAP}}(\mathcal{D})] &= \mathbb{E}\left[\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^\top (\mathbf{X}\boldsymbol{\omega} + \boldsymbol{\varepsilon})\right] \\ &= \mathbb{E}\left[\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right)^{-1} (\mathbf{X}^\top \mathbf{X})\boldsymbol{\omega}\right] \\ &\neq \boldsymbol{\omega}.\end{aligned}$$

As $\lambda \to 0$, the MAP solution becomes closer and closer to being unbiased. The covariance is

$$\mathrm{Cov}[\mathbf{w}_{\mathrm{MAP}}(\mathcal{D})] = \sigma^2 \mathbb{E}\left[(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1}\right].$$

This covariance is much less susceptible to ill-conditioned $\mathbf{X}^\top \mathbf{X}$, because as discussed above, the shift by λ improves the condition. Consequently, we expect $\mathbf{w}_{\mathrm{MAP}}$ to have lower variance across different datasets that could have been observed. This correspondingly implies that we are less likely to overfit to anyone dataset. Notice that as $\lambda \to \infty$, the variance decreases to zero, but the bias increases to infinity. As depicted in Figure 5.5, there is an optimal choice of λ that minimizes this bias-variance trade-off—if we could find it.

Exercise: Derive the covariance formula for $\mathbf{w}_{\mathrm{MAP}}(\mathcal{D})$.

The bias-variance trade-off comes in many forms. One such trade-off is in the selection of our function class. If we select a simple function class, the class is likely not large enough—not powerful enough—to represent the true function. This introduces some bias, but likely also has lower variance, because that simpler function class is less likely to overfit to any one dataset. If this class is too simple, we might say that our function is *underparametrized* and is *underfitting*. On the other hand, if we select a more powerful function class, that

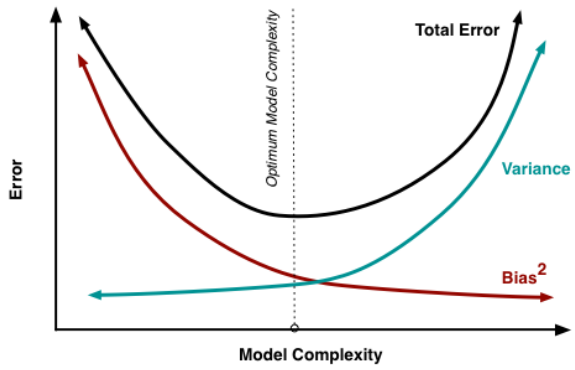


Figure 5.5: The bias-variance trade-off. Image obtained from: http://scott.fortmann-roe.com/docs/BiasVariance.html

does contain the true function, we may not have any bias but could have high variance due to the ability to find a function in your large class that overfits a given dataset. In this setting, we might say the function is over-parametrized, and though we have the ability to learn a highly accurate function, it will be difficult to actually find that function amongst this larger class. Instead, one is likely to select a model that overfits to the given data, and does not generalize to new data (i.e., performs poorly on new data).

Finding the balance between bias and variance, and between underfitting and overfitting, is a core problem in machine learning. We discuss ways to theoretically and empirically investigate this trade-off, in Chapter 10.

Remark: Above we assumed that the true model was linear, and so the only bias introduced was from the regularization. This assumed that the hypothesis space of linear functions was sufficiently powerful. In reality, when using linear regression with regularization, we are introducing bias both from selecting a simpler function class and from the regularization. If a powerful basis is used to first transform the data, to provide nonlinear functions even though the solution uses linear regression, then it is feasible that this function class is sufficiently powerful, and the bias is mostly due to regularization.

Chapter 6

More advanced optimization principles

Given the optimization background in Chapter 2, and seeing how it is useful in the following chapters, we can now turn to more advanced optimization approaches. We will now discuss more in-depth how we obtain the second-order gradient descent update for the multivariate case. We then discuss some computational improvements on these methods, particularly through the use of improved step-size selection techniques, by using stochastic gradient descent and some small modifications to deal with non-differentiable points. Finally, we will also provide some basics on constrained optimization. When moving to the multivariate case, it will be useful to get used to multivariate calculus. We provide some basic rules in Section B.1; a more complete reference for these rules can be found in the (highly useful) matrix cookbook [16].

6.1 Multivariate gradient descent

We can generalize the discussion on obtaining the gradient descent update in Section 2.2 from the univariate case to the multivariate case using the multivariate Taylor series approximation. The second-order Taylor approximation for a real-valued function of multiple variables can be written as

$$c(\mathbf{w}) \approx \hat{c}(\mathbf{w}) = c(\mathbf{w}_0) + \nabla c(\mathbf{w}_0)^\top (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_0)^\top \mathbf{H}_{c(\mathbf{w}_0)} (\mathbf{w} - \mathbf{w}_0),$$

where

$$\nabla c(\mathbf{w}_0) = \left(\frac{\partial c}{\partial w_1}(\mathbf{w}_0), \frac{\partial c}{\partial w_2}(\mathbf{w}_0), ..., \frac{\partial c}{\partial w_d}(\mathbf{w}_0) \right) \qquad \in \mathbb{R}^d$$

is the gradient of function c evaluated at $\mathbf{w}_0$ and

$$\mathbf{H}_{c(\mathbf{w}_0)} = \begin{bmatrix} \frac{\partial^2 c}{\partial w_1^2}(\mathbf{w}_0) & \frac{\partial^2 c}{\partial w_1 \partial w_2}(\mathbf{w}_0) & \cdots & \frac{\partial^2 c}{\partial w_1 \partial w_d}(\mathbf{w}_0) \\ \frac{\partial^2 c}{\partial w_2 \partial w_1}(\mathbf{w}_0) & \frac{\partial^2 c}{\partial w_2^2}(\mathbf{w}_0) & & \\ \vdots & \vdots & \ddots & \\ \frac{\partial^2 c}{\partial w_d \partial w_1}(\mathbf{w}_0) & \cdots & & \frac{\partial^2 c}{\partial w_d^2}(\mathbf{w}_0) \end{bmatrix} \qquad \in \mathbb{R}^{d \times d}$$

is the Hessian matrix of function c evaluated at $\mathbf{w}_0$. We provide some intuition for the Hessian in the next section, but here it can be intuitively considered analogous to the second derivative. Like the second derivative, it provides information about the curvature of the function, and so provides useful information about how much to step in the direction of the gradient for each w_i.

As a reminder about matrix-vector multiplication, the product of a $d \times d$ matrix $\mathbf{H}$ and $d \times 1$ vector $\mathbf{w}$ is a $d \times 1$ vector $\mathbf{Hw}$. Then, taking $\mathbf{w}^\top \mathbf{Hw}$ is the dot product between a $1 \times d$ vector $\mathbf{w}^\top$ and $d \times 1$ vector $\mathbf{Hw}$, resulting in a scalar. For matrix-vector multiplication,

$$\mathbf{Hw} = \begin{bmatrix} \mathbf{H}_{1:} \\ \mathbf{H}_{2:} \\ \vdots \\ \mathbf{H}_{d:} \end{bmatrix} \mathbf{w} = \begin{bmatrix} \mathbf{H}_{1:}\mathbf{w} \\ \mathbf{H}_{2:}\mathbf{w} \\ \vdots \\ \mathbf{H}_{d:}\mathbf{w} \end{bmatrix} = \begin{bmatrix} \langle \mathbf{H}_{1:}, \mathbf{w} \rangle \\ \langle \mathbf{H}_{2:}, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{H}_{d:}, \mathbf{w} \rangle \end{bmatrix}$$

When performing matrix-vector multiplication, you can just imagine the vector $\mathbf{w}$ turning sideways and multiplying each row of $\mathbf{H}$. For matrix-matrix multiplication, $\mathbf{AB}$, you have to ensure that the second dimension of $\mathbf{A}$ equals the first dimension of $\mathbf{B}$. The matrix-matrix multiplication decomposes into matrix-vector multiplication, for each column of $\mathbf{B}$.

As before, to get the incremental update, we can take the gradient of this approximation and obtain the (local) stationary point. Using the basic rules summarized below in Section B.1, the gradient of $\hat{c}(\mathbf{w})$ is

$$\nabla \hat{c}(\mathbf{w}) = \nabla c(\mathbf{w}_0) + \mathbf{H}_{c(\mathbf{w}_0)} (\mathbf{w} - \mathbf{w}_0) .$$

Again, we want to find $\mathbf{w}_1$ such that this gradient is zero. If you are not yet familiar with the inverse of a matrix, this will be discussed more in later sections of these notes (particularly for linear regression in Chapter 5). For now, to solve for $\mathbf{H}_{c(\mathbf{w}_0)} (\mathbf{w} - \mathbf{w}_0) = -\nabla c(\mathbf{w}_0)$, one can compute the inverse $\mathbf{H}^{-1}_{c(\mathbf{w}_0)}$ and multiply both sides of the equation by this inverse. This is again analogous to the inverse of a scalar: $h^{-1}h = 1$. The corresponding multivariate update, extended beyond Equation (2.1) for the scalar case, is

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \left(\mathbf{H}_{c(\mathbf{w}_i)}\right)^{-1} \nabla c(\mathbf{w}_i). \tag{6.1}$$

In Equation 6.1, both gradient and Hessian are evaluated at point $\mathbf{w}_i$.

The size of the Hessian makes the choice between first-order and second-order gradient descent less obvious in the multivariate case. Unlike the scalar setting, computing the Hessian itself is expensive (quadratic in the size of $\mathbf{w}$) and it is further even more expensive to compute the inverse of the Hessian. For this reason, more light-weight first-order updates are often preferred. For example, if computing the Hessian costs $O(d^2 n)$ as it does for the linear regression objective, then the computational complexity of the second-order gradient descent is $O(d^3 + d^2 n)$ in each iteration, assuming $O(d^3)$ time for finding matrix inverses. On the other hand, again for linear regression, the computational complexity for first-order gradient descent is only $O(dn)$ per iteration.

The first order update for the multivariate case is an even greater approximation, because the whole Hessian is approximated with a scalar $\frac{1}{\eta}$ (making the Hessian approximation a diagonal matrix with $\frac{1}{\eta}$ on the diagonal). The gradient of the first-order approximation then becomes

$$\nabla \hat{c}(\mathbf{w}) = \nabla c(\mathbf{w}_0) + \frac{1}{\eta} (\mathbf{w} - \mathbf{w}_0)$$

and the resulting first-order update is

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i \nabla c(\mathbf{w}_i). \tag{6.2}$$

The selection of this step-size is an important consideration. We have already discussed one basic strategy to select the step-size; in Section 6.5, we discuss a few more.

6.2 Properties of the Hessian

Like the second-derivative, the Hessian reflects the curvature of the function at the point $\mathbf{w}_0$. Each entry reflects how the partial derivative for w_j changes when w_i is changed.

For additional intuition, consider the directional derivative. The directional derivative reflects how a (multivariate) function changes when stepping a small amount t in some fixed direction $\mathbf{u}$

$$\lim_{t \to 0} \frac{c(\mathbf{w} + t\mathbf{u}) - c(\mathbf{w})}{t}.$$

Once we restrict ourselves to how the function changes in this one direction, it is easier to imagine and it allows us to use the familiar second derivative rules for the univariate setting. Let

$$\begin{aligned} \mathbf{w}(t) &= \mathbf{w} + t\mathbf{u} \\ g(t) &= c(\mathbf{w}(t)). \end{aligned}$$

We can use the chain rule on $g(t)$ to compute the derivative w.r.t. t.

$$\begin{aligned} g'(t) &= \nabla c(\mathbf{w}(t))^\top \frac{\partial(\mathbf{w}(t))}{\partial t} \\ &= \nabla c(\mathbf{w}(t))^\top \mathbf{u} \\ g'(0) &= \nabla c(\mathbf{w}(0))^\top \mathbf{u} \\ &= \nabla c(\mathbf{w})^\top \mathbf{u} = 0 \end{aligned}$$

where the last equality occurs because $\mathbf{w}$ is a stationary point and so $\nabla c(\mathbf{w}) = \mathbf{0}$. The second derivative is

$$\begin{aligned} g''(t) &= \frac{\partial(\mathbf{w}(t))}{\partial t}^\top \mathbf{H}_{c(\mathbf{w}(t))} \frac{\partial(\mathbf{w}(t))}{\partial t} \\ &= \mathbf{u}^\top \mathbf{H}_{c(\mathbf{w}(t))} \mathbf{u} \\ g''(0) &= \mathbf{u}^\top \mathbf{H}_{c(\mathbf{w})} \mathbf{u} \end{aligned}$$

For this stationary point $\mathbf{w}$ (corresponding to $t = 0$) to be a local minimum, $g''(0)$ has to satisfy the second derivative test: $g''(0) > 0$. This test is only satisfied if $\mathbf{H}_{c(\mathbf{w})}$ is positive definite, by definition of a positive definite matrix. Recall that a positive-definite matrix $\mathbf{H}$ is one for which, given any $\mathbf{u} \neq \mathbf{0}$, $\mathbf{u}^\top \mathbf{H} \mathbf{u} > 0$, or equivalently, has all eigenvalues greater than zero. Since $\mathbf{u}$ was an arbitrary direction away from $\mathbf{w}$, the Hessian must be positive-definite to ensure that $g''(0) > 0$ for all $\mathbf{u} \neq \mathbf{0}$.

The eigenvalues of the Hessian, therefore, reflect the curvature of the function locally. If $\mathbf{H}_{c(\mathbf{w})}$ has a very small eigenvalue λ, then the corresponding eigenvector $\mathbf{u}$—satisfying $\mathbf{H}_{c(\mathbf{w})}\mathbf{u} = \lambda\mathbf{u}$—is a direction away fro $\mathbf{w}$ where the function is almost flat. This is because $g''(0) = \mathbf{u}^\top \mathbf{H}_{c(\mathbf{w})} \mathbf{u} = \lambda \|\mathbf{u}\|_2^2 = \lambda$ is very small.

Example 16: We can now consider the Hessian $H_{c(\mathbf{w})}$ for the linear regression solution. This Hessian will enable us to verify if we indeed found a local minimum, of if instead we found a stationary point that is a local maximum or a saddle point. The Hessian is

$$H_{c(\mathbf{w})} = 2\mathbf{X}^\top \mathbf{X}.$$

This Hessian is positive semi-definite matrix. To see why, consider that for any vector $\mathbf{w} \neq \mathbf{0}$,

$$\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} = (\mathbf{X}\mathbf{w})^\top \mathbf{X}\mathbf{w} = \|\mathbf{X}\mathbf{w}\|_2^2 \geq 0$$

where equality can only happen—for some $\mathbf{w}$—if the columns of $\mathbf{X}$ are linearly dependent. Since the Hessian is positive semi-definite for every $\mathbf{w}$, this verifies the convexity of $c(\mathbf{w})$. Furthermore, if the columns of $\mathbf{x}$ are linearly independent, the Hessian is positive definite, which implies that the global minimum is unique. □

6.3 Handling big data sets

One common approach to handling big datasets is to use *stochastic approximation*, where samples are processed incrementally. To see how this would be done, let us revisit the gradient of the objective function, $\nabla c(\mathbf{w})$. We obtained a closed form solution for $\nabla c(\mathbf{w}) = \mathbf{0}$; however, for many other objective functions, solving for $\nabla c(\mathbf{w}) = \mathbf{0}$ in a closed form way is not possible. Instead, we start at some initial $\mathbf{w}_0$ (typically random), and then step in the direction of the negative of the gradient until we reach a local minimum. This approach is called *gradient descent* and is summarized in Algorithm 2. Notice that here the gradient is normalized by the number of samples n, as $\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$ grows with the number of samples and makes it more difficult to select the stepsize.

Algorithm 2: Batch Gradient Descent$(c, \mathbf{X}, \mathbf{y})$

```
// A non-optimized, basic implementation of batch gradient descent
w ← random vector in R^d
err ← ∞
tolerance ← 10e^{-4}
max iterations ← 10e^5
while |c(w) − err| > tolerance and have not reached max iterations do
    err ← c(w)                          ▷ for linear regression, c(w) = 1/(2n) ||Xw − y||_2^2
    g ← ∇c(w)                           ▷ for linear regression, ∇c(w) = 1/n X^T (Xw − y)
    // The step-size η could be chosen by line-search, as in Algorithm 1
    η ← line search(w, c, g)
    w ← w − ηg
return w
```

For a large number of samples n, however, computing the gradient across all samples can be expensive or infeasible. An alternative is to approximate the gradient less accurately with fewer samples. In stochastic approximation, we typically approximate the gradient with one sample[1], as in Algorithm 3. Though this approach may appear to be too much of an approximation, there is a long theoretical and empirical history indicating its effectiveness (see for example [6, 5]). With ever increasing data-set size for many scenarios, the generality of stochastic approximation makes it arguably the modern approach to dealing with big data. For specialized scenarios, there are of course other approaches. For one example, see [17].

[1] Mini-batches are a way to obtain a better approximation but remain efficient.

The training algorithm for stochastic gradient descent can now be revised to randomly draw one data point at a time from $\mathcal{D}$ and then update the current weights using the previous equation. Typically, in practice, this entails iterating one or more times over the dataset in order (assuming it is random, with i.i.d. samples). Each iteration over the dataset is called an *epoch*. The conditions for convergence typically include conditions on the step-sizes, requiring them to decrease over time. As with batch gradient descent, these stochastic gradient descent updates will converge, though with more oscillation around the true weight vector, with the decreasing step-size progressively smoothing out these oscillations.

Algorithm 3: Stochastic Gradient Descent$(c, \mathbf{X}, \mathbf{y})$

$\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
for $i = 1, \ldots$ number of epochs **do**
  Shuffle data points from $1, \ldots, n$
  for $j = 1, \ldots, n$ **do**
    $\mathbf{g} \leftarrow \nabla c_j(\mathbf{w})$ ▷ for linear regression, $\nabla c_j(\mathbf{w}) = (\mathbf{x}_j^\top \mathbf{w} - y_j)\mathbf{x}_j$
    // For convergence, the step-size η_t needs to decrease with time, such as
    // $\eta_t = \eta_0 t^{-1/2}$ or $\eta_t = \eta_0 i^{-1}$ for an initial η_0 (e.g., $\eta_0 = 1.0$).
    // In practice, it is common to pick a fixed, small stepsize
    $\eta_t \leftarrow i^{-1}$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \mathbf{g}$
return $\mathbf{w}$

6.4 Non-smooth but still continuous optimization

We assume throughout these notes that our objectives are continuous. However, this need not mean that they are smooth: in some cases, these continuous objectives may have non-differentiable points. For example, the ℓ_1 regularizer is non-differentiable at 0, making $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$ non-differentiable. One strategy is to use sub-gradient descent; loosely, this amounts to selecting a reasonable choice for the gradient at the non-differentiable point. Here, for example, we could take the partial derivative of ℓ_1 for w_j to be zero at zero, -1 for $w_j < 0$ and 1 for $w_j > 0$. Unfortunately, this descent is slow because there is a tendency to jump around zero. Unlike ℓ_2, the gradient does not gradually decrease near zero, slowly decreasing w_j, but rather jumps between two large values -1 and 1. With such large gradient, it is difficult to gradually decrease w_j to zero, even if that is the optimal solution.

One alternative for such non-smooth objectives is to use proximal methods. The idea is simple: use gradient descent for the smooth component of the optimization (the error term $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$), and then for values in $\mathbf{w}$ that are close to zero, set them to zero. This thresholding idea, though simple, is a theoretically sound approach for optimizing with the non-smooth ℓ_1. This thresholding operator is called the proximal operator, and can be seen as a projection operator. Each time $\mathbf{w}$ is updated with the gradient, it moves it away from a sparse solution; the proximal operator then projects $\mathbf{w}$ back onto the space of sparse solutions. The proximal operator for ℓ_1 is applied element-wise to $\mathbf{w}$, and so is defined on

each $\mathbf{w}_i$ as, with stepsize η and regularization parameter λ,

$$\text{prox}_{\eta\lambda\ell_1}(\mathbf{w}_i) = \begin{cases} \mathbf{w}_i - \eta\lambda & \text{if } \mathbf{w}_i > \eta\lambda \\ 0 & \text{if } |\mathbf{w}_i| \leq \eta\lambda \\ \mathbf{w}_i + \eta\lambda & \text{if } \mathbf{w}_i < -\eta\lambda. \end{cases}$$

The proximal operator on the entire vector $\mathbf{w}$ is defined element-wise: $\text{prox}_{\eta\lambda\ell_1}(\mathbf{w}) = [\text{prox}_{\eta\lambda\ell_1}(\mathbf{w}_1), \ldots, \text{prox}_{\eta\lambda\ell_1}(\mathbf{w}_d)]$. Nicely, the theory states that the stepsize should be no larger than the inverse of the Lipschitz constant for the smooth part of the objective, where intuitively the Lipschitz constants reflects how quickly the function changes. In Algorithm 4, we provide a gradient descent algorithm for the incremental update with the ℓ_1 regularizer, introduced as an algorithm called ISTA [4]. More generally, proximal methods are used for other non-smooth objectives, though in these notes we only consider Lasso.

Algorithm 4: Batch gradient descent for ℓ_1 regularized linear regression $(\mathbf{X}, \mathbf{y}, \lambda)$

$\mathbf{w} \leftarrow \mathbf{0} \in \mathbb{R}^d$
err $\leftarrow \infty$
tolerance $\leftarrow 10e^{-4}$
// Precomputing these matrices, to avoid recomputing them in the loop
$XX \leftarrow \frac{1}{n}\mathbf{X}^\top\mathbf{X}$
$Xy \leftarrow \frac{1}{n}\mathbf{X}^\top\mathbf{y}$
// This stepsize is specific to the least-squares loss for linear regression
$\eta \leftarrow 1/(2\|XX\|_F)$
while $|c(\mathbf{w}) - \text{err}| >$ tolerance and have not reached max iterations **do**
    err $\leftarrow c(\mathbf{w})$
    // Proximal operator projects back into the space of sparse solutions given by ℓ_1
    $\mathbf{w} \leftarrow \text{prox}_{\eta\lambda\ell_1}(\mathbf{w} - \eta XX\mathbf{w} + \eta Xy)$
return $\mathbf{w}$

6.5 More methods to select the step-size

Because selecting the step-size is such an important part of an effective descent algorithm, there are many ways to do so. In addition to line search, one of the most popular methods is to use quasi-second-order (or quasi-Newton) methods. As we saw, the inverse of the Hessian provides a good way to select the stepsize, but is typically too expensive to compute let alone invert. Quasi-second-order methods approximate the Hessian, with as little storage and computation as possible. One of the simplest such approximations is to approximate only the diagonal of the Hessian, and the invert it, which only costs $O(d)$ computation and space. Such an approximation is typically quite poor for even the diagonal of the inverse Hessian, and so is not commonly used. Instead, the most popular methods include LBFGS [14], Adadelta [21] and Adam [11].

Chapter 7

Generalized Linear Models

In previous sections, we saw that the statistical framework provided valuable insights into linear regression, especially with respect to explicitly stating most of the assumptions in the system (we will see the full picture only when Bayesian formulation is used). These assumptions were necessary to rigorously estimate parameters of the model, which could then be subsequently used for prediction on previously unseen data points.

In this section, we introduce generalized linear models (GLMs) which extend ordinary least-squares regression beyond Gaussian probability distributions and linear dependencies between the features and the target. This generalization will also introduce you to a broader range of loss functions, called Bregman divergences.

We shall first revisit the main points of the ordinary least-squares regression. There, we assumed that a set of i.i.d. data points with their targets $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ were drawn according to some distribution $p(\mathbf{x}, y)$. We also assumed that an underlying relationship between the features and the target was linear, i.e.

$$Y = \sum_{j=0}^{d} \omega_j X_j + \varepsilon,$$

where $\boldsymbol{\omega}$ was a set of unknown weights and ε was a zero-mean normally distributed random variable with variance σ^2. In order to simplify generalization, we will slightly reformulate this model. In particular, it will be useful to separate the underlying linear relationship between the features and the target from the fact that Y was normally distributed. That is, we will write that

1. $\mathbb{E}[y|\mathbf{x}] = \boldsymbol{\omega}^\top \mathbf{x}$
2. $p(y|\mathbf{x}) = \mathcal{N}(\mu, \sigma^2)$

with $\mu = \boldsymbol{\omega}^\top \mathbf{x}$ connecting the two expressions. This way of formulating linear regression will allow us (*i*) to generalize the framework to non-linear relationships between the features and the target as well as (*ii*) to use the error distributions other than Gaussian.

7.1 Exponential transfer and the Poisson distribution

We will start first with an example of a GLM, before moving on to the general class and general definition. Assume that data points correspond to cities in the world—described by some numerical features—and that the target variable is the number of sunny days observed in a particular year. The target variable y may look like a Poisson distribution, given features $\mathbf{x}$. It would be more natural, therefore, to model $p(y|\mathbf{x}) = \text{Poisson}(\lambda)$, where

$\lambda > 0$ is the parameter (mean) of the Poisson distribution: $\mathbb{E}[y|\mathbf{x}] = \lambda$. However, because $\lambda \in \mathbb{R}^+$, it would not be appropriate to model λ with $\boldsymbol{\omega}^\top \mathbf{x} \in \mathbb{R}$. Rather, we would like to transfer our linear prediction with some function f to adjusts the range of the linear combination of features to the domain of the parameters of the probability distribution.

We can do so by introducing an exponential transfer for this Poisson distribution, and more generically, later any invertible transfer function f. If we can instead estimate $\boldsymbol{\omega}$ such that $\lambda = e^{\boldsymbol{\omega}^\top \mathbf{x}}$, then we can guarantee our estimates are in the correct range. Alternatively, one can consider that we are learning a linear weighting of features to learn a transformed parameter, $\log(\lambda) = \boldsymbol{\omega}^\top \mathbf{x}$. This simple modification is why these models are called *generalized* linear models, because the key component is a still a linear weighting. We formalize the types of distributions and transfers that can be considered in the below sections, but first finish off this example with Poisson regression to provide a concrete example.

To establish the GLM model for Poisson regression, we assume (1) an exponential transfer between the expectation of the target and linear combination of features, and (2) the Poisson distribution for the target variable.

1. $\mathbb{E}[y|\mathbf{x}] = \exp\left(\boldsymbol{\omega}^\top \mathbf{x}\right)$ or $\log(\mathbb{E}[y|\mathbf{x}]) = \boldsymbol{\omega}^\top \mathbf{x}$
2. $p(y|\mathbf{x}) = \text{Poisson}(\lambda)$

Exploiting the fact that $E\left[y|\mathbf{x}\right] = \lambda$, we connect the two formulas using $\lambda = e^{\boldsymbol{\omega}^\top \mathbf{x}}$. The resulting probability distribution is

$$p(y|\mathbf{x}) = \frac{e^{\boldsymbol{\omega}^\top \mathbf{x} y} \cdot e^{-e^{\boldsymbol{\omega}^\top \mathbf{x}}}}{y!}$$

for any $y \in \mathbb{N}$.

We can use maximum likelihood estimation to find the parameters of the regression model. The log-likelihood function has the form

$$\begin{aligned} ll(\mathbf{w}) &= \sum_{i=1}^{n} ll_i(\mathbf{w}) \\ ll_i(\mathbf{w}) &= \mathbf{w}^\top \mathbf{x}_i y_i - e^{\mathbf{w}^\top \mathbf{x}_i} - \ln y_i! \end{aligned}$$

Our goal is to minimize the negative log-likelihood: $\min_{\mathbf{w}} -ll(\mathbf{w})$. It is easy to see that $\nabla ll(\mathbf{w}) = \mathbf{0}$ does not have a closed-form solution. Therefore, unlike linear regression, we will have to use gradient descent. We could choose to use first-order or second-order gradient descent, and batch or stochastic gradient descent. The key step in any of these is to first compute the gradient for one sample. We start by deriving the partial derivative of the negative log-likelihood for one sample

$$\begin{aligned} -\frac{\partial ll_i(\mathbf{w})}{\partial w_j} &= e^{\mathbf{w}^\top \mathbf{x}_i} x_{ij} - x_{ij} y_i \\ &= x_{ij}\left(e^{\mathbf{w}^\top \mathbf{x}_i} - y_i\right). \end{aligned}$$

The gradient for one sample is

$$-\nabla ll_i(\mathbf{w}) = \mathbf{x}_i \cdot (p_i - y_i)$$

where $p_i = e^{\mathbf{w}^\top \mathbf{x}_i}$ is the prediction. Notice that $p_i - y_i$ corresponds to a prediction error, for sample i. The batch gradient is

$$\begin{aligned} -\nabla ll(\mathbf{w}) &= -\sum_{i=1}^{n} \nabla ll_i(\mathbf{w}) \\ &= \sum_{i=1}^{n} \mathbf{x}_i \left(p_i - y_i\right) \\ &= \mathbf{X}^\top \left(\mathbf{p} - \mathbf{y}\right) \end{aligned} \tag{7.1}$$

where $\mathbf{p}$ is a vector with elements $p_i = e^{\mathbf{w}^\top \mathbf{x}_i}$, where $\mathbf{p} - \mathbf{y}$ is an error vector.

Commonly, one would now just do stochastic or batch gradient descent. For stochastic gradient descent, each step consists of using the gradient for one sample (i.e., $-ll_i(\mathbf{w}_t)$) and for batch gradient descent, each step consists of using the gradient for all samples (i.e., $-ll(\mathbf{w}_t)$). We can additionally consider the Hessian matrix, both to evaluate the properties of the stationary points as well as to allow for second-order gradient descent—though it is likely too expensive if d is large. The second partial derivative of the negative log likelihood function for one sample is

$$\begin{aligned} -\frac{\partial^2 ll_i(\mathbf{w})}{\partial w_j \partial w_k} &= x_{ij} e^{\mathbf{w}^\top \mathbf{x}_i} x_{ik} \\ &= x_{ij} p_i x_{ik} \end{aligned}$$

with

$$-\frac{\partial^2 ll(\mathbf{w})}{\partial w_j \partial w_k} = -\sum_{i=1}^{n} \frac{\partial^2 ll_i(\mathbf{w})}{\partial w_j \partial w_k}.$$

For $\mathbf{P}$ an $n \times n$ diagonal matrix with p_i on the diagonal, the Hessian matrix is therefore

$$H_{-ll(\mathbf{w})} = -\mathbf{X}^\top \mathbf{P} \mathbf{X}. \tag{7.2}$$

This matrix is positive definite if $\mathbf{X}$ is not low-rank, which would mean there is only one stationary point and that it is the global minimum. In fact, we know that the objective for Poisson regression is convex, even if $\mathbf{X}$ is not full rank, and so all stationary points are global minima.

Exercise: Is there one global minimum if $\mathbf{X}$ is not full rank?

Exercise: What is the second-order update for Poisson regression?

7.2 Exponential family distributions

In the previous section, we used a specific example to illustrate how to generalize beyond Gaussian distributions. The approach more generally extends to any exponential family distribution. For simplicity, here we focus on the natural exponential family, which is sufficient for most generalized linear models. The natural exponential family is a class of probability distributions with the following form

$$p(x|\theta) = \exp\left(\theta x - a(\theta) + b(x)\right)$$

where $\theta \in \mathbb{R}$ is the parameter to the distribution, $a : \mathbb{R} \to \mathbb{R}$ is a log-normalizer function and $b : \mathbb{R} \to \mathbb{R}$ is a function of only x that will typically be ignored in our optimization because it is not a function of θ. Many of the often encountered (families of) distributions are members of the exponential family; e.g. exponential, Gaussian, Gamma, Poisson, or the binomial distributions. Therefore, it is useful to generically study the exponential family to better understand commonalities and differences between individual member functions.

Example 17: The Poisson distribution can be expressed as

$$p(x|\lambda) = \exp\left(x \log \lambda - \lambda - \log x!\right),$$

where $\lambda \in \mathbb{R}^+$ and $\mathcal{X} = \mathbb{N}_0$. Thus, $\theta = \log \lambda$, $a(\theta) = e^\theta$, and $b(x) = -\log x!$. □

Now let us get some further insight into the properties of the exponential family parameters and why this class is convenient for estimation. The function $a(\theta)$ is typically called the log-partitioning function or simply a log-normalizer. It is called this because

$$a(\theta) = \log \int_{\mathcal{X}} \exp\left(\theta x + b(x)\right) dx$$

and so plays the role of ensuring that we have a valid density: $\int_{\mathcal{X}} p(x)dx = 1$. Importantly, for many common GLMs, the derivative of a corresponds to the inverse of the link function. For example, for Poisson regression, the link function $g(\theta) = \log(\theta)$, and the derivative of a is e^θ, which is the inverse of g. Therefore, as we discuss below, the log-normalizer for an exponential family informs what link g should be used (or correspondingly the transfer $f = g^{-1}$).

The properties of this log-normalizer are also key for estimation of generalized linear models. It can be derived that

$$\begin{aligned} \frac{\partial a(\theta)}{\partial \theta} &= \mathbb{E}\left[X\right] \\ \frac{\partial^2 a(\theta)}{\partial \theta^2} &= \mathrm{V}[X] \end{aligned}$$

7.3 Formalizing generalized linear models

We shall now formalize the generalized linear models. The two key components of GLMs can be expressed as

1. $\mathbb{E}[y|\mathbf{x}] = f(\boldsymbol{\omega}^\top \mathbf{x})$ or $g(\mathbb{E}[y|\mathbf{x}]) = \boldsymbol{\omega}^\top \mathbf{x}$ where $g = f^{-1}$
2. $p(y|\mathbf{x})$ is an Exponential Family distribution

The function f is called the transfer function and g is called the link function. For Poisson regression, f is the exponential function, and as we shall see for logistic regression, f is the sigmoid function. The transfer function adjusts the range of $\boldsymbol{\omega}^\top \mathbf{x}$ to the domain of Y; because of this relationship, link functions are usually not selected independently of the distribution for Y. The generalization to the exponential family from the Gaussian distribution used in ordinary least-squares regression, allows us to model a much wider range of target functions. GLMs include three widely used models: linear regression, Poisson regression and logistic regression, which we will talk about in the next chapter about classification.

To relate these more clearly to exponential family distributions, we have to consider conditional distributions. Each $p(y|\mathbf{x})$ is an exponential family distribution, with parameter $\theta = \mathbf{x}^\top \mathbf{w}$. When learning $\mathbf{w}$—by maximizing likelihood—we are learning the parameter θ_i for each sample $(\mathbf{x}_i, y_i)$. The general negative log-likelihood is

$$\begin{aligned} -ll(\mathbf{w}) &= -\log \prod_{i=1}^{n} e^{\theta_i y_i - a(\theta) + b(y_i)} \\ &= -\sum_i \left(\theta_i y_i - a(\theta) + b(y_i)\right) \\ &= -\sum_i ll_i(\mathbf{w}) \end{aligned}$$

with gradients

$$\begin{aligned} -\frac{\partial ll_i(\mathbf{w})}{\partial w_j} &= \frac{\partial a(\theta_i)}{\partial w_j} - \frac{\partial \theta_i}{\partial w_j} y_i \\ &= \frac{\partial a(\theta_i)}{\partial \theta_i}\frac{\partial \theta_i}{\partial w_j} - \frac{\partial \theta_i}{\partial w_j} y_i \\ &= \left(\frac{\partial a(\theta_i)}{\partial \theta_i} - y_i\right)\frac{\partial \theta_i}{\partial w_j}. \end{aligned}$$

As was clear for Poisson regression, there is no guarantee of a closed-form solution for $\mathbf{w}$. Therefore, GLM formulations usually use iterative techniques, like gradient descent. Hence, a single mechanism can be used for a wide range of link functions and probability distributions, using these above gradients.

This update can be made more concrete, using the most common setting for GLMs. Importantly, this setting only requires knowledge of the transfer function f, without explicitly needing to know the log-normalized a. This simplification arises from the connection between the transfer f and the log-normalizer a alluded to above. We have discussed that the transfer function f is chosen to reflect the range of the output variable y. However, the choice should have other properties as well. In particular, we would like to ensure that the g provides a smooth, convex negative log-likelihood, to simplify optimization. Usefully, the parameter a of the exponential family distribution provides us with just such a choice: $f = \nabla a$. Because $\frac{\partial \theta_i}{\partial w_j} = x_{ij}$ for $\theta_i = \mathbf{x}_i^\top \mathbf{w}$, we get that

$$\begin{aligned} -\frac{\partial ll_i(\mathbf{w})}{\partial w_j} &= \left(\frac{\partial a(\theta_i)}{\partial \theta_i} - y_i\right)\frac{\partial \theta_i}{\partial w_j} \\ &= \left(f(\theta_i) - y_i\right) x_{ij} \\ &= \left(f(\mathbf{x}_i^\top \mathbf{w}) - y_i\right) x_{ij} \end{aligned}$$

Therefore, given the appropriate transfer f for the desired exponential family distribution, the stochastic gradient descent update is simply

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left(f(\mathbf{x}_i^\top \mathbf{w}_t) - y_i\right) \mathbf{x}_i$$

and the batch gradient descent update is

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \sum_{i=1}^{n} \left(f(\mathbf{x}_i^\top \mathbf{w}_t) - y_i \right) \mathbf{x}_i \\ &= \mathbf{w}_t - \eta_t \mathbf{X}^\top (\mathbf{p} - \mathbf{y})\end{aligned}$$

where $\mathbf{p}_i = f(\mathbf{x}_i^\top \mathbf{w}_t)$. To examine the Hessian, the second partial derivative of the negative log likelihood function for one sample is

$$-\frac{\partial^2 ll_i(\mathbf{w})}{\partial w_j \partial w_k} = x_{ij} \frac{\partial f(\theta_i)}{\partial \theta_i} x_{ik}.$$

For $\mathbf{D}$ an $n \times n$ diagonal matrix with $\frac{\partial f(\theta_i)}{\partial \theta_i}$ on the diagonal, the Hessian matrix is therefore

$$H_{-ll(\mathbf{w})} = -\mathbf{X}^\top \mathbf{D} \mathbf{X}. \tag{7.3}$$

As in Poisson regression, this matrix is guaranteed to be positive semi-definite, and further positive definite if $\mathbf{X}$ is not low-rank.

Remark: The common setting of $f = \nabla a$ for GLMs has a connection to widely used objectives called *Bregman divergences*. These divergences are written as $D_a(\hat{y}||y)$, indicating the difference between $\hat{y}$ and y, where the divergence is parametrized by a. The minimization of this Bregman divergence corresponds to the minimization of the negative log-likelihood of the corresponding natural exponential family:

$$\operatorname*{argmin}_\theta D_a(x||g(\theta)) = \operatorname*{argmin}_\theta - \ln p(x|\theta).$$

See [20, Section 2.2] and [2] for more details about this relationship.

Note that the chosen link does not necessarily have to correspond to the derivative of a. Rather, this provides a mechanism for ensuring a nice loss function, since Bregman divergences have nice properties, including being convex in the first argument. However, this does not mean that any other link will necessarily result in an undesirable loss function.

Chapter 8

Linear Classifiers

Suppose we are interested in building a linear classifer $f : \mathbb{R}^d \to \{-1, +1\}$. Linear classifiers try to find the relationship between inputs and outputs by constructing a linear function (a point, a line, a plane or a hyperplane) that splits $\mathbb{R}^d$ into two half-spaces. The two half-spaces act as decision regions for the positive and negative examples, respectively. Given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ consisting of positive and negative examples, there are many ways in which linear classifiers can be constructed. For example, a training algorithm may explicitly work to position the decision surface in order to separate positive and negative examples according to some problem-relevant criteria; e.g. it may try to minimize the fraction of examples on the incorrect side of the decision surface. Alternatively, the goal of the training algorithm may be to directly estimate the posterior distribution $p(y|\mathbf{x})$, in which case the algorithm is more likely to rely on the formal parameter estimation principles; e.g. it may maximize the likelihood. An example of a classifier with a linear decision surface is shown in Figure 8.1.

To simplify the formalism in the following sections, we will add a component $x_0 = 1$ to each input $(x_1, \ldots, x_d)$. This extends the input space to $\mathcal{X} = \mathbb{R}^{d+1}$ but, fortunately, it also leads us to a simplified notation in which the decision boundary in $\mathbb{R}^d$ can be written as $\mathbf{w}^\top \mathbf{x} = 0$, where $\mathbf{w} = (w_0, w_1, \ldots, w_d)$ is a set of weights and $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_d)$ is any element of the input space. Neverthless, we should remember that the actual inputs are d-dimensional.

Earlier in the introductory remarks, we presented a classifier as a function $f : \mathcal{X} \to \mathcal{Y}$ and have transformed the learning problem into approximating $p(y|\mathbf{x})$. In the case of

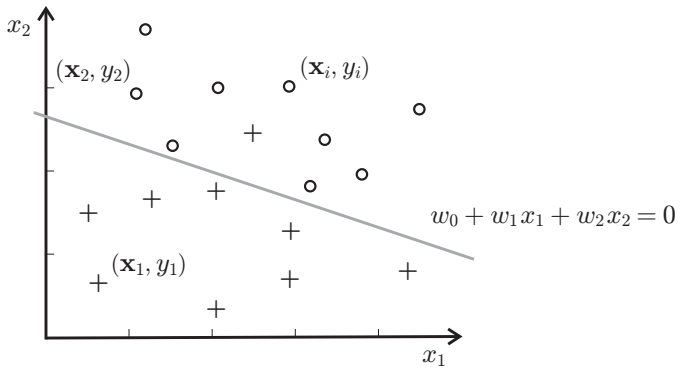


Figure 8.1: A data set in $\mathbb{R}^2$ consisting of nine positive and nine negative examples. The gray line represents a linear decision surface in $\mathbb{R}^2$. The decision surface does not perfectly separate positives from negatives.

linear classifiers, our flexibility is restricted because our method must learn the posterior probabilities $p(y|\mathbf{x})$ and at the same time have a linear decision surface in $\mathbb{R}^d$. This, however, can be achieved if $p(y|\mathbf{x})$ is modeled as a monotonic function of $\mathbf{w}^\top\mathbf{x}$; e.g. $\tanh(\mathbf{w}^\top\mathbf{x})$ or $(1+e^{-\mathbf{w}^\top\mathbf{x}})^{-1}$. Of course, a model trained to learn posterior probabilities $p(y|\mathbf{x})$ can be seen as a function $g : \mathcal{X} \to [0, 1]$. Then, the conversion from g to f is a straightforward application of the maximum a posteriori principle: the predicted output is positive if $g(\mathbf{x}) \geq 0.5$ and negative if $g(\mathbf{x}) < 0.5$.

8.1 Logistic regression

Let us consider binary classification in $\mathbb{R}^d$, where $\mathcal{X} = \mathbb{R}^{d+1}$ and $\mathcal{Y} = \{0, 1\}$. Logistic regression is a Generalized Linear Model, where the distribution over Y given $\mathbf{x}$ is a Bernoulli distribution, and the transfer function is the sigmoid function, also called the logistic function,

$$\sigma(t) = \left(1 + e^{-t}\right)^{-1}$$

plotted in Figure 8.2. In the same terminology as for GLMs, the transfer function is the sigmoid and the link function—the inverse of the transfer function— is the logit function $\text{logit}(x) = \ln \frac{x}{1-x}$, with

1. $E[y|\mathbf{x}] = \sigma(\boldsymbol{\omega}^\top\mathbf{x})$
2. $p(y|\mathbf{x}) = \text{Bernoulli}(\alpha)$ with $\alpha = E[y|\mathbf{x}]$.

The Bernoulli distribution, with α a function of $\mathbf{x}$, is

$$\begin{aligned} p(y|\mathbf{x}) &= \begin{cases} \left(\frac{1}{1+e^{-\boldsymbol{\omega}^\top\mathbf{x}}}\right)^y & \text{for } y = 1 \\ \left(1 - \frac{1}{1+e^{-\boldsymbol{\omega}^\top\mathbf{x}}}\right)^{1-y} & \text{for } y = 0 \end{cases} \\ &= \sigma(\mathbf{x}^\top\mathbf{w})^y(1 - \sigma(\mathbf{x}^\top\mathbf{w}))^{1-y} \end{aligned} \tag{8.1}$$

where $\boldsymbol{\omega} = (\omega_0, \omega_1, \ldots, \omega_d)$ is a set of unknown coefficients we want to recover (or learn). Notice that our prediction is $\sigma(\boldsymbol{\omega}^\top\mathbf{x})$, and that it satisfies

$$p(y = 1|\mathbf{x}) = \sigma(\boldsymbol{\omega}^\top\mathbf{x}).$$

Therefore, as with many binary classification approaches, our goal is to predict the probability that the class is 1; given this probability, we can infer $p(y = 0|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$.

8.1.1 Predicting class labels

The function learned by logistic regression returns a probability, rather than an explicit prediction of 0 or 1. Therefore, we have to take this probability estimate and convert it to a suitable prediction of the class. For a previously unseen data point $\mathbf{x}$ and a set of learned coefficients $\mathbf{w}$, we simply calculate the posterior probability as

$$P(Y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\top\mathbf{x}}}.$$

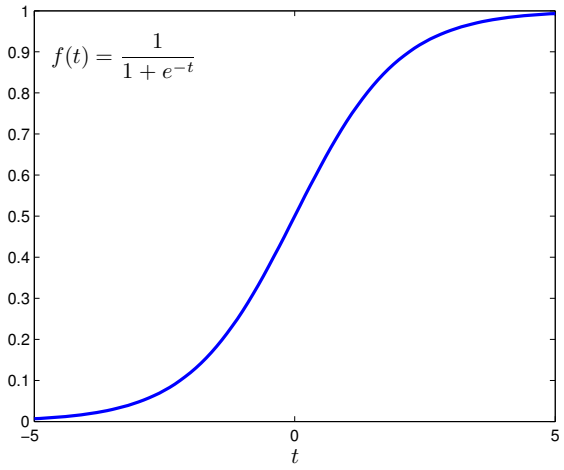


Figure 8.2: Sigmoid function in $[-5, 5]$ *interval.*

If $P(Y = 1|\mathbf{x}, \mathbf{w}) \geq 0.5$ we conclude that data point $\mathbf{x}$ should be labeled as positive ($\hat{y} = 1$). Otherwise, if $P(Y = 1|\mathbf{x}, \mathbf{w}^*) < 0.5$, we label the data point as negative ($\hat{y} = 0$). The predictor maps a $(d+1)$-dimensional vector $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_d)$ into a zero or one.

As we discuss in Chapter 10, however, this threshold need not be 0.5. In some cases, one might care more about failing to identify a positive (e.g., failing to identify a disease); in such a case, it may be safer to err on the side of a smaller threshold, so that more instances are labeled as positive. Further, the probability values themselves can be reflective: even if both classifiers produce good accuracies, it is preferable to have a classifier that consistently produces probabilities near 0.9 and 0.1, rather than less confident probabilities that hover around 0.5. This reason for this is that small perturbations are expected to have more impact on the second classifier, which could suddenly erroneously swap the labeling on an instance. To quantify this aspect of the prediction, other measures—such as the reported operating curve—are preferred, which we discuss in Section 10.4. For now, we will assume this simpler thresholding and leave more advanced evaluation of classification algorithms to later.

Notice that the logistic regression classifier is a linear classifier, despite the fact that the sigmoid is non-linear. This is because $P(Y = 1|\mathbf{x}, \mathbf{w}) \geq 0.5$ only when $\mathbf{w}^\top \mathbf{x} \geq 0$. The expression $\mathbf{w}^\top \mathbf{x} = 0$ represents the equation of a hyperplane that separates positive and negative examples.

8.1.2 Maximum likelihood estimation for logistic regression

Because logistic regression is a GLM, we can use the generic gradient descent algorithm derived in Section 7.3 with transfer $f = \sigma$, with the following gradient of the negative log-likelihood per sample

$$-\frac{\partial ll_i(\mathbf{w})}{\partial w_j} = \left(\sigma(\mathbf{x}_i^\top \mathbf{w}) - y_i\right) x_{ij}.$$

Even though we know the final update, as an exercise, we will explicitly derive the maximum likelihood solution. As before, assume that the data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is an

i.i.d. sample from a fixed but unknown probability distribution $p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x})$. The data is generated by randomly drawing a point $\mathbf{x}$ according to $p(\mathbf{x})$ and then sets its class label Y according to the Bernoulli distribution in (8.1). The negative log-likelihood for this dataset is $-ll(\mathbf{w}) = \sum_{i=1}^{n} -ll_i(\mathbf{w})$ where

$$\begin{aligned} ll_i(\mathbf{w}) &= \log p(y_i|\mathbf{x}) \\ &= y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)\right) \\ &= \left(y_i \log \left(\frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}}\right) + (1 - y_i) \log \left(1 - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}}\right)\right). \end{aligned}$$

The negative of the log-likelihood with this ll_i is typically referred to as the *cross-entropy*.

From here, you could take the derivative of each component in this sum, using the chain rule for the sigmoid. For the first component, with $p_i = \sigma(\theta_i)$,

$$\begin{aligned} \frac{\partial y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i)}{\partial w_j} &= y_i \frac{\partial \log \sigma(\theta_i)}{\partial \theta_i} \frac{\partial \theta_i}{w_j} \\ &= y_i \frac{\partial \log p_i}{\partial p_i} \frac{\partial p_i}{\partial \theta_i} \frac{\partial \theta_i}{w_j} \\ &= y_i \frac{1}{p_i} \frac{\partial p_i}{\partial \theta_i} \frac{\partial \theta_i}{w_j} \\ &= y_i \frac{1}{p_i} \sigma(\theta_i)(1 - \sigma(\theta_i)) x_{ij} \\ &= y_i (1 - \sigma(\theta_i)) x_{ij} \end{aligned}$$

because

$$\frac{\partial \sigma(\theta_i)}{\partial \theta_i} = \sigma(\theta_i)(1 - \sigma(\theta_i)).$$

You can verify this step for yourself, but explicitly plugging in the definition of σ. For the second component, following similar steps, we get

$$\frac{\partial (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))}{\partial w_j} = (y_i - 1)\sigma(\theta_i) x_{ij}$$

Summing these together and taking the negative, we end up with the gradient $(p_i - y_i)x_{ij}$.

For further practice taking gradients of these objectives, we could have slightly rearranged the objective before taking the gradient. This would lead to another path to derive the update rule for logistic regression, which we go through now. Notice first that

$$\left(1 - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}}\right) = \frac{e^{-\mathbf{w}^\top \mathbf{x}_i}}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}}$$

giving

$$\begin{aligned} ll(\mathbf{w}) &= \sum_{i=1}^{n} \Big(- y_i \cdot \log \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right) + (1 - y_i) \cdot \log \left(e^{-\mathbf{w}^\top \mathbf{x}_i}\right) - (1 - y_i) \cdot \log \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right) \Big) \\ &= \sum_{i=1}^{n} \left((y_i - 1)\, \mathbf{w}^\top \mathbf{x}_i + \log \left(\frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}}\right)\right). \end{aligned}$$

Again, unlike linear regression, its clear there is no closed-form solution to $\nabla ll(\mathbf{w}) = \mathbf{0}$. Thus, we have to proceed with iterative optimization methods. We initialize $\mathbf{w}_0$ usually to a random vector, or potentially with the linear regression solution which provides a much better initial point. Because the objective is convex, the initialization only affects the number of steps, but should not prevent the gradient descent from converging to a global minimum. The stochastic gradient descent update is

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left(\sigma(\mathbf{x}_i^\top \mathbf{w}_t) - y_i\right) \mathbf{x}_i$$

and the batch gradient descent update is

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \sum_{i=1}^{n} \left(\sigma(\mathbf{x}_i^\top \mathbf{w}_t) - y_i\right) \mathbf{x}_i \\ &= \mathbf{w}_t - \eta_t \mathbf{X}^\top \left(\sigma(\mathbf{X}\mathbf{w}_t) - \mathbf{y}\right)\end{aligned}$$

where we overload the definition of σ when applied to a vector to mean that it individually applies to each element in that vector: $\sigma(\mathbf{v}) = [\sigma(v_1), \ldots, \sigma(v_n)]$. The Hessian is $H_{-ll(\mathbf{w})} = -\mathbf{X}^\top \mathbf{D}\mathbf{X}$ where $\mathbf{D}$ is a $n \times n$ diagonal matrix with $p_i(1-p_i)$ on the diagonal (see (7.3)).

Weighted conditional likelihood function

In certain situations, it may be justified to allow for unequal importance of each data point. This modifies the conditional likelihood function to

$$l(\mathbf{w}) = \prod_{i=1}^{n} p_i^{c_i y_i} \cdot (1-p_i)^{c_i(1-y_i)},$$

where $0 \leq c_i \leq 1$ is a cost for data point i. Taking that $\mathbf{C} = \operatorname{diag}(c_1, c_2, \ldots, c_n)$ we can now express the gradient of the negative log-likelihood as

$$-\nabla ll(\mathbf{w}) = \mathbf{X}^\top \mathbf{C}\left(\mathbf{p} - \mathbf{y}\right)$$

and the Hessian as

$$H_{-ll(\mathbf{w})} = -\mathbf{X}^\top \mathbf{C}\mathbf{P}\left(\mathbf{I} - \mathbf{P}\right)\mathbf{X}.$$

It is interesting to observe that the Hessian remains positive semi-definite. Thus, the update rule is expected to converge to a global minimum.

8.1.3 Issues with minimizing Euclidean distance

A natural question is why we went down this route for linear classification. Instead of explicitly assuming $P(Y = 1|\mathbf{x}, \mathbf{w})$ is a Bernoulli distribution and computing the maximum likelihood solution for $\sigma(\mathbf{x}^\top \mathbf{w}) = E[Y|\mathbf{x}] = P(Y = 1|\mathbf{x}, \mathbf{w})$, we could have simply decided to use $\sigma(\mathbf{x}^\top \mathbf{w})$ to predict targets $y \in \{0, 1\}$ and then tried to minimize their difference, using our favorite loss (the squared loss). Unfortunately, this more haphazard problem specification results in a non-convex optimization. In fact, there is a result that using the Euclidean error for the sigmoid transfer gives exponentially many local minima in the

number of features [1]. For interest about this alternate route, we will show that this direction leads to a non-convex optimization.

Let the error function with Euclidean distance now be written as

$$\text{Err}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - p_i)^2 \qquad \triangleright\, p_i = \sigma(\mathbf{x}_i^\top \mathbf{w}), \quad e_i = y_i - p_i$$

The minimization of $\text{Err}(\mathbf{w})$ is formally expressed as

$$\begin{aligned} \mathbf{w}^* &= \arg\min_{\mathbf{w}} \{\text{Err}(\mathbf{w})\} \\ &= \arg\min_{\mathbf{w}} \left\{ \sum_{i=1}^{n} (y_i - p_i)^2 \right\}. \end{aligned} \tag{8.2}$$

Similar to the maximum likelihood process, our goal will be to calculate the gradient vector and the Hessian of the error function. The partial derivatives of the error function can be calculated as follows

$$\begin{aligned} \frac{\partial \text{Err}(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j} \sum_{i=1}^{n} e_i^2 \\ &= \sum_{i=1}^{n} 2 \cdot e_i \cdot \frac{\partial e_i}{\partial w_j} \\ &= 2 \cdot \sum_{i=1}^{n} \left(y_i - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}} \right) \cdot \frac{1}{(1 + e^{-\mathbf{w}^\top \mathbf{x}_i})^2} \cdot e^{-\mathbf{w}^\top \mathbf{x}_i} \cdot (-x_{ij}) \\ &= -2 \cdot \sum_{i=1}^{n} x_{ij} \cdot \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}} \cdot \left(1 - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}} \right) \cdot \left(y_i - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}} \right) \\ &= -2 \mathbf{f}_j^\top \mathbf{P} \left(\mathbf{I} - \mathbf{P} \right) \left(\mathbf{y} - \mathbf{p} \right). \end{aligned}$$

This provides the gradient vector in the following form

$$\nabla \text{Err}(\mathbf{w}) = -2 \mathbf{X}^\top \mathbf{P} \left(\mathbf{I} - \mathbf{P} \right) \left(\mathbf{y} - \mathbf{p} \right).$$

Matrix $\mathbf{J} = \mathbf{P} \left(\mathbf{I} - \mathbf{P} \right) \mathbf{X}$ is referred to as Jacobian. In general, Jacobian is an $n \times d$ matrix calculated as

$$J_{\text{Err}(\mathbf{w})} = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_d} \\ \vdots & \ddots & & \\ & & & \\ \frac{\partial e_n}{\partial w_1} & & & \frac{\partial e_n}{\partial w_d} \end{bmatrix}.$$

The second partial derivative of the error function can be found as

$$\begin{aligned} \frac{\partial^2 \text{Err}(\mathbf{w})}{\partial w_j \partial w_d} &= 2 \cdot \sum_{i=1}^{n} \frac{\partial e_i}{\partial w_d} \cdot \frac{\partial e_i}{\partial w_j} + e_i \cdot \frac{\partial^2 e_i}{\partial w_j \partial w_d} \\ &= 2 \cdot \sum_{i=1}^{n} x_{ij} \cdot \Big(p_i^2 (1 - p_i)^2 + p_i \cdot (1 - p_i) \cdot (2 p_i - 1) \cdot (y_i - p_i) \Big) \cdot x_{ik}. \end{aligned}$$

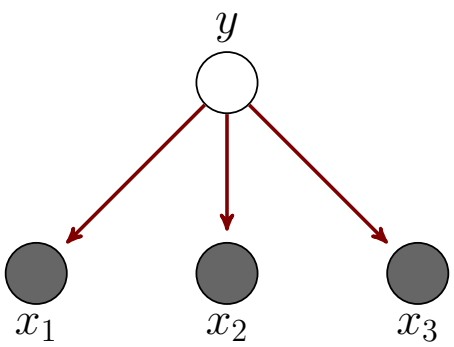


Figure 8.3: Naive Bayes graphical model, with three features.

Thus, the Hessian can be computed as

$$\begin{aligned} H_{\text{Err}(\mathbf{w})} &= 2\mathbf{X}^\top (\mathbf{I}-\mathbf{P})^\top \mathbf{P}^\top \mathbf{P} (\mathbf{I}-\mathbf{P}) \mathbf{X} + 2\mathbf{X}^\top (\mathbf{I}-\mathbf{P})^\top \mathbf{P}^\top \mathbf{E}(2\mathbf{P}-\mathbf{I})\mathbf{X} \\ &= 2\mathbf{J}^\top \mathbf{J} + 2\mathbf{J}^\top \mathbf{E}(2\mathbf{P}-\mathbf{I})\mathbf{X}, \end{aligned}$$

where $\mathbf{P} = \text{diag}\{\mathbf{p}\}$, $\mathbf{E} = \text{diag}\{\mathbf{e}\}$ is a diagonal matrix containing elements $E_{ii} = e_i = y_i - p_i$ and $\mathbf{I}$ is an identity matrix.

We can now see that the Hessian is not guaranteed to be positive semi-definite. This means that $\text{Err}(\mathbf{w})$ is not convex, i.e. it must have multiple minima with different values of the objective function. Finding a global optimum depends on how favorable the initial solution $\mathbf{w}^{(0)}$ is and how well the weight update step can escape local minima to find better ones. Minimization of this non-convex function, however, will be much more problematic than the convex cross-entropy.

8.2 Naive Bayes Classifier

Naive Bayes classification is a generative approach to prediction. So far, we have discussed discriminative approaches (linear regression, logistic regression), which attempt to learn $p(y|\mathbf{x})$. For a generative setting, we learn $p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y)$. As you can imagine, this can be a more difficult undertaking, as we also need to learn the distribution over the features themselves. For naive Bayes, we significantly simply learning this joint distribution by making a strong assumption: the features are conditionally independent given the label. This assumption is demonstrated by the graphical model in Figure 8.3.

As with discriminative classifiers, like logistic regression, the decision rule for labeling a point as class c (i.e., $y = 1$) is

$$\underset{c \in \{1,\ldots,k}{\text{argmax}}\, p(y = c|\mathbf{x})$$

where $p(y = 1|\mathbf{x}) \propto p(\mathbf{x}|y = 1)p(y = 1)$. For two classes, this corresponds to picking class 1 if $p(y = 1|\mathbf{x}) \geq p(y = 0|\mathbf{x})$.

To start, we will assume a simpler setting with binary features, and then address continuous features. Note that naive Bayes is a linear classifier for binary features; more generally, however, it is not necessarily a linear classifier. Note that a linear classifier is one in which the two classes are separated by a linear plane, i.e., the decision boundary is according to some linear combination of features.

8.2.1 Binary features and linear classification

Let $\mathcal{D} = \{(\mathbf{x}_i, y\}_{i=1}^n$ be an input data set, where $\mathcal{X} = \{0,1\}^d$ and $\mathcal{Y} = \{0,1\}$. Under the naive Bayes assumption, the features are independent given the class label. Therefore, we can write

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y).$$

A suitable choice for this simpler univariate probabilities is a Bernoulli distribution, since each x_j is binary, giving

$$p(x_j|y=c) = p_{j,c}^{x_j}(1-p_{j,c})^{1-x_j}$$

The parameters for the Bernoulli distributions are $p_{j,c} = p(x_j = 1|y = c)$, with a different parameter $p_{j,c}$ for each class value c and for each feature. We can easily learn this parameter from data by calculating

$$p_{j,c} = \frac{\text{number of times } x_j = 1 \text{ for class } c}{\text{number of datapoints labeled as class } c}.$$

Similarly, we can learn the prior $p_c = p(y = c)$ using

$$p_c = p(y=c) = \frac{\text{number of datapoint labeled as class } c}{\text{total number of datapoints}}.$$

Notice that this approach could also be accomplished for more classes than just two. The prediction on a new point $\mathbf{x}$ is then

$$\begin{aligned}
\max_{c\in\mathcal{Y}} p(y=c|\mathbf{x}) &= \max_{c\in\mathcal{Y}} p(\mathbf{x}|y=c)p(y=c) \\
&= \max_{c\in\mathcal{Y}} \prod_{j=1}^{d} p(x_j|y=c)p(y=c) \\
&= \max_{c\in\mathcal{Y}} \prod_{j=1}^{d} p_{j,c}p_c
\end{aligned}$$

Exercise: The solution above is intuitive, and comes from similarly deriving the maximum likelihood solution. Assuming that you have n datapoints and the chosen distribution $p(x_i|y)$ is Bernoulli as described above, derive the maximum likelihood parameters $p_{j,c}, p_c$ for $j = 1, \ldots, d, c = 0, 1$. To make things simpler, use the log of the likelihood.

Linear classification boundary with binary features and targets

Interestingly, naive Bayes classifier with binary features and two classes is a linear classifier. This is somewhat surprising, as this generative approach looks very different from what we did before. To see why this is the case, notice that the classifier will make a positive decision when

$$p(y=1|\mathbf{x}) \geq p(y=0|\mathbf{x})$$

that is, when

$$p(\mathbf{x}|y=1)p(y=1) \geq p(\mathbf{x}|y=0)p(y=0)$$

We will shorten this notation using $p(y=0) = p(0)$, $p(\mathbf{x}|y=0) = p(\mathbf{x}|0)$, etc. Using the naive Bayes assumption, we now have

$$p(1)\prod_{j=1}^{d} p(x_j|1) \geq p(0)\prod_{j=1}^{d} p(x_j|0)$$

which, after applying a logarithm, then becomes

$$\log p(1) + \sum_{j=1}^{d} \log p(x_j|1) \geq \log p(0) + \sum_{j=1}^{d} \log p(x_j|0)$$

Let us now investigate class-conditional probabilities $p(x_j|y)$, when $y \in \{0,1\}$. Recall that each feature is Bernoulli distributed, i.e.

$$p(x_j|1) = p_{j,1}^{x_j}(1-p_{j,1})^{1-x_j}$$

and

$$p(x_j|0) = p_{j,0}^{x_j}(1-p_{j,0})^{1-x_j}$$

where parameters $p_{j,c}$ are estimated from the training set. Taking $p(y=c) = p_c$, we have

$$\sum_{j=1}^{d} x_j \log \frac{p_{j,1}(1-p_{j,0})}{(1-p_{j,1})p_{j,0}} + \sum_{j=1}^{d} \log \frac{1-p_{j,1}}{1-p_{j,0}} + \log \frac{p_1}{p_0} \geq 0$$

We can write the previous expression as

$$w_0 + \sum_{j=1}^{d} w_j x_j \geq 0$$

where

$$w_0 = \log \frac{p_1}{p_0} + \sum_{j=1}^{d} \log \frac{1-p_{j,1}}{1-p_{j,0}}$$

$$w_j = \log \frac{p_{j,1}(1-p_{j,0})}{(1-p_{j,1})p_{j,0}} \qquad j \in \{1,2,\ldots,d\}$$

Therefore, in the case of binary features, naive Bayes is a linear classifier.

8.2.2 Continuous naive Bayes

For continuous features, a Bernoulli distribution is no longer appropriate for $p(x_j|y)$ and we need to choose a different conditional distribution $p(\mathbf{x}|y)$. A common choice is a Gaussian distribution, now with a different mean and variance for each feature and class, $\mu_{j,c}, \sigma^2_{j,c}$:

$$p(x_j|y=c) = (2\pi\sigma^2_{j,c})^{-1/2} \exp\left(-\frac{(x_j-\mu_{j,c})^2}{2\sigma^2_{j,c}}\right).$$

Since y is still discrete, we can approximate $p(y)$ using counts as before. The maximum likelihood mean and variance parameters correspond to the sample mean and sample covariance for each given class separately. This involves computing the mean and variance of feature j across the datapoints labeled with class c:

$$\mu_{j,c} = \frac{\sum_{i=1}^{n} 1(y_i = c)x_j}{\text{number of datapoints labeled as class } c}$$
$$\sigma^2_{j,c} = \frac{\sum_{i=1}^{n} 1(y_i = c)(x_j - \mu_{j,c})^2}{\text{number of datapoints labeled as class } c}$$

Exercise: Derive the maximum likelihood formulation for a Gaussian naive Bayes model, and check that the solution does in fact match the sample mean and variance for each feature and class separately, as above.

8.3 Multinomial logistic regression

Now let us consider discriminative multiclass classification, where $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, 2, \ldots, k\}$. This setting arises naturally in machine learning, where there is often more than two categories. For example, if we want to predict the blood type (A, B, AB and O) of an individual, then we have four classes. Here we discuss multiclass classification where we only want to label a datapoint with one class out of k. In other settings, one might want to label a datapoint with multiple classes; this is briefly mentioned at the end of this section.

We can nicely generalize to this setting using the idea of multinomials and the corresponding link function, as with the other generalized linear models. The multinomial distribution is a member of the exponential family. We can write

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{y_1! \ldots y_k!} p(y_1 = 1|\mathbf{x})^{y_1} \ldots, p(y_k = 1|\mathbf{x})^{y_k} \tag{8.3}$$

where the usual numerator $n! = 1$ because $n = \sum_{j=1}^{k} y_j = 1$ since we can only have one class value. As with logistic regression, we can parametrize $p(y_j = 1|\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}_j)$. However, we must also ensure that $\sum_{j=1}^{k} p(y_j = 1|\mathbf{x}) = 1$. To do so, we "pivot" around the final class, $p(y_k = 1|\mathbf{x}) = 1 - \sum_{j=1}^{k-1} p(y_j = 1|\mathbf{x})$ and only explicitly learn $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$. Note that these models are not learned independently, because they are tied by the probability for the last class. The parameters can be represented as a matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$ where $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_k]$ is composed of k weight vectors with $\mathbf{w}_k = \mathbf{0}$. We will see why we fix $\mathbf{w}_k = \mathbf{0}$.

The transfer (inverse of the link) for this setting is the softmax transfer

$$\begin{aligned}\text{softmax}(\mathbf{x}^\top \mathbf{W}) &= \left[\frac{\exp(\mathbf{x}^\top \mathbf{w}_1)}{\sum_{j=1}^{k} \exp(\mathbf{x}^\top \mathbf{w}_j)}, \ldots, \frac{\exp(\mathbf{x}^\top \mathbf{w}_k)}{\sum_{j=1}^{k} \exp(\mathbf{x}^\top \mathbf{w}_j)}\right] \\ &= \left[\frac{\exp(\mathbf{x}^\top \mathbf{w}_1)}{\mathbf{1}^\top \exp(\mathbf{x}^\top \mathbf{W})}, \ldots, \frac{\exp(\mathbf{x}^\top \mathbf{w}_k)}{\mathbf{1}^\top \exp(\mathbf{x}^\top \mathbf{W})}\right]\end{aligned}$$

and the prediction is $\text{softmax}(\mathbf{x}) = \hat{\mathbf{y}} \in [0, 1]^k$, which gives the probability in each entry of being labeled as that class, where $\hat{\mathbf{y}}^\top \mathbf{1} = 1$ signifying that the probabilities sum to 1. Note that this model encompasses the binary setting for logistic regression, because $\sigma(\mathbf{x}^\top \mathbf{w}) =$

$(1+\exp(-\mathbf{x}^\top \mathbf{w}))^{-1} = \frac{\exp(\mathbf{x}^\top \mathbf{w})}{1+\exp(\mathbf{x}^\top \mathbf{w})}$. The weights for multinomial logistic regression with two classes are then $\mathbf{W} = [\mathbf{w},\ \mathbf{0}]$ giving

$$\begin{aligned} p(y=0|\mathbf{x}) &= \frac{\exp(\mathbf{x}^\top \mathbf{w})}{\mathbf{1}^\top \exp(\mathbf{x}^\top \mathbf{W})} \\ &= \frac{\exp(\mathbf{x}^\top \mathbf{w})}{\exp(\mathbf{x}^\top \mathbf{w}) + \exp(\mathbf{x}^\top \mathbf{0})} \\ &= \frac{\exp(\mathbf{x}^\top \mathbf{w})}{\exp(\mathbf{x}^\top \mathbf{w}) + 1} \\ &= \sigma(\mathbf{x}^\top \mathbf{w}). \end{aligned}$$

Similarly, for $k > 2$, by fixing $\mathbf{w}_k = \mathbf{0}$, the other weights $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$ are learned to ensure that $p(y=k|\mathbf{x}) = \frac{\exp(\mathbf{x}^\top \mathbf{w}_k)}{\mathbf{1}^\top \exp(\mathbf{x}^\top \mathbf{W})} = \frac{1}{1+\sum_{j=1}^{k-1} \exp(\mathbf{x}^\top \mathbf{w}_j)}$ and that $\sum_{j=1}^{k} p(y=j|\mathbf{x}) = 1$.

With the parameters of the model parameterized by $\mathbf{W}$ and the softmax transfer, we can determine the maximum likelihood formulation. By plugging in the parameterization into Equation (8.3), taking the negative log of that likelihood and dropping constants, we arrive at the following loss for samples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$

$$\min_{\mathbf{W} \in \mathbb{R}^{d \times k} : \mathbf{W}_{:k} = \mathbf{0}} \sum_{i=1}^{n} \log\left(\mathbf{1}^\top \exp(\mathbf{x}_i^\top \mathbf{W})\right) - \mathbf{x}_i^\top \mathbf{W} \mathbf{y}_i$$

with gradient

$$\nabla \sum_{i=1}^{n} \left(\log\left(\mathbf{1}^\top \exp(\mathbf{x}_i^\top \mathbf{W})\right) - \mathbf{x}_i^\top \mathbf{W} \mathbf{y}_i\right) = \sum_{i=1}^{n} \frac{\exp(\mathbf{x}_i^\top \mathbf{W})^\top \mathbf{x}_i^\top}{\mathbf{1}^\top \exp(\mathbf{x}_i^\top \mathbf{W})} - \mathbf{x}_i \mathbf{y}_i^\top.$$

As before, we do not have a closed form solution for this gradient, and will use iterative methods to solve for $\mathbf{W}$. Note that here, unlike previous methods, we have a constraint on part of the variable. However, this was solely written this way for convenience. We do not optimize $\mathbf{W}_{:k}$, as it is fixed at zero; one can rewrite this minimization and gradient to only apply to the $W_{:(1:k-1)}$. This corresponds to initializing $\mathbf{W}_{:k} = \mathbf{0}$, and then only using the first $k-1$ columns of the gradient in the update to $W_{:(1:k-1)}$.

The final prediction $\text{softmax}(\mathbf{x}^\top \mathbf{W}) \in [0,1]$ gives the probabilities of being in a class. As with logistic regression, to pick one class, the highest probability value is chosen. For example, with $k = 4$, we might predict $[0.1 \ \ 0.2 \ \ 0.6 \ \ 0.1]$ and so decide to classify the point into class 3.

Remark about overlapping classes: If you want to predict multiple classes for a datapoint $\mathbf{x}$, then a common strategy is to learn separate binary predictors for each class. Each predictor is queried separately, and a datapoint will label each class as 0 or 1, with potentially more than one class having a 1. Above, we examined the case where the datapoint was exclusively in one of the provided classes, by setting $n = 1$ in the multinomial.

Chapter 9

Representations for machine learning

At first, it might seem that the applicability of linear regression and classification to real-life problems is greatly limited. After all, it is not clear whether it is realistic (most of the time) to assume that the target variable is a linear combination of features. Fortunately, the applicability of linear regression is broader than originally thought. The main idea is to apply a non-linear transformation to the data matrix $\mathbf{x}$ prior to the fitting step, which then enables a non-linear fit. Obtaining such a useful feature representation is a central problem in machine learning.

We will first examine fixed representations for linear regression: polynomial curve fitting and radial basis function (RBF) networks. Then, we will discuss learning representations.

9.1 Radial basis function networks and kernel representations

The idea of radial basis function (RBF) networks is a natural generalization of the polynomial curve fitting and approaches from the previous Section. Given data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we start by picking p points to serve as the "centers" in the input space $\mathcal{X}$. We denote those centers as $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_p$. Usually, these can be selected from $\mathcal{D}$ or computed using some clustering technique (e.g. the EM algorithm, K-means).

When the clusters are determined using a Gaussian mixture model, the basis functions can be selected as

$$\phi_j(\mathbf{x}) \quad = \quad e^{-\frac{1}{2}(\mathbf{x}-\mathbf{c}_j)^T \Sigma_j^{-1} (\mathbf{x}-\mathbf{c}_j)},$$

where the cluster centers and the covariance matrix are found during clustering. When K-means or other clustering is used, we can use

$$\phi_j(\mathbf{x}) = e^{-\frac{\left\|\mathbf{x}-\mathbf{c}_j\right\|^2}{2\sigma_j^2}},$$

where σ_j's can be separately optimized; e.g. using a validation set. In the context of multidimensional transformations from $\mathbf{x}$ to $\mathbf{\Phi}$, the basis functions can also be referred to as *kernel functions*, i.e. $\phi_j(\mathbf{x}) = k_j(\mathbf{x}, \mathbf{c}_j)$. Matrix

$$\mathbf{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_p(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & & \\ \vdots & & \ddots & \\ \phi_0(\mathbf{x}_n) & & & \phi_p(\mathbf{x}_n) \end{bmatrix}$$

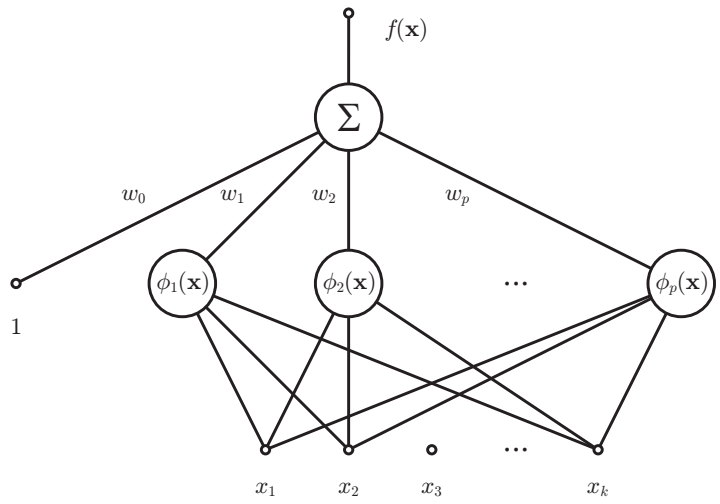


Figure 9.1: Radial basis function network.

is now used as a new data matrix. For a given input $\mathbf{x}$, the prediction of the target y will be calculated as

$$\begin{aligned} f(\mathbf{x}) &= w_0 + \sum_{j=1}^{p} w_j \phi_j(\mathbf{x}) \\ &= \sum_{j=0}^{p} w_j \phi_j(\mathbf{x}) \end{aligned}$$

where $\phi_0(\mathbf{x}) = 1$ and $\mathbf{w}$ is to be found. It can be proved that with a sufficiently large number of radial basis functions we can accurately approximate any function. As seen in Figure 9.1, we can think of RBFs as neural networks.

RBF networks and kernel representations are highly related. The main distinction is that kernel representations use any kernel function for the similarity measure $k(\mathbf{x}, \mathbf{c}_j) = \phi_j(\mathbf{x})$, where radial basis functions are one example of a kernel. In addition, if an RBF kernel is chosen, for kernel representations typical the centers are selected from the training dataset. For RBF networks, the selection of the centers is left generally as an important step, where they can be selected from the training set but can also be selected in other ways.

9.2 Learning representations

There are many approaches to learning representations. Two dominant approaches are (semi-supervised) matrix factorization techniques and neural networks. Neural networks build on the generalized linear models we have discussed, stacking multiple generalized linear models together. Matrix factorization techniques (e.g., dimensionality reduction, sparse coding) typically factorize the input data into a dictionary and a new representation (a basis). We will first discuss neural networks, and then discuss the many unsupervised and semisupervised learning techniques that are encompassed by matrix factorizations.

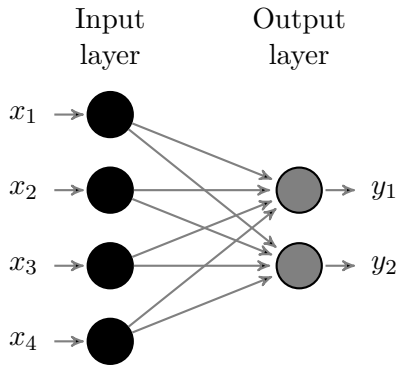


Figure 9.2: Generalized linear model, such as logistic regression.

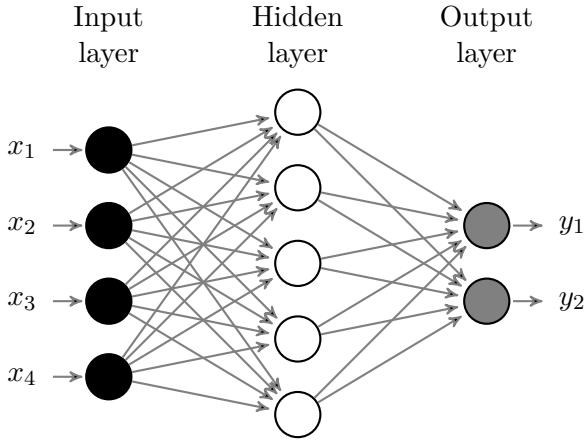


Figure 9.3: Standard two-layer neural network.

9.2.1 Neural networks

Neural networks are a form of supervised representation learning. As before, the goal is to learn a function of inputs, f, to produce a prediction of the target: $f(\mathbf{x})$. The addition of hidden layers, with non-linear activation functions, enables learning of nonlinear functions f. For some intuition, one can consider that all the first hidden layers constitute representation layer, with learning on the last layer corresponding to supervised prediction part. Figure 9.2 shows the graphical model for the generalized linear models we discussed in the previous chapters, where the weights and corresponding transfer can be thought of as being on the arrows (as they are not random variables). Figure 9.3 shows a neural network with one hidden-layer; this is called a two-layer neural network, as there are two layers of weights.

In the figure, the neural network inputs a 4-dimensional feature vector $\mathbf{x} = [x_1,\ x_2,\ x_3,\ x_4]$ (i.e., $d = 4$) and outputs a 2-dimensional prediction $\mathbf{y} = [y_1,\ y_2]$ (i.e., $m = 2$). The hidden layer consists of a mapping from $\mathbf{x}$ to a new representation that is 5-dimensional (i.e., $k_1 = 5$ as per the notation below). For the neural network, let each node in this hidden representation be indexed by $k \in \{1, \ldots, 5\}$. Each h_k consists of a transformation of a linear weighting of $\mathbf{x}$, such as a sigmoid transfer: $h_k = \sigma\left(\sum_{j=1}^{d} x_j w_{kj}\right) = \sigma\left(\mathbf{x}\mathbf{w}_k\right)$ where $\mathbf{w}_k \in \mathbb{R}^d$ is the weights on the first layer used to produce the kth node in the hidden representation.

Example 18: For a simple example, consider $d = 1$ (i.e., one input observation), $m = 1$ (i.e., one output), $k_1 = 2$ (i.e., 2-dimensional hidden layer) and a sigmoid transfer to get the first hidden layer. Assume we are given one instance (x, y). Then input observation x is transformed into

$$\mathbf{h} = [h_1,\ h_2], \qquad \text{with } h_1 = \sigma(x w_1^{(2)}) \text{ and } h_2 = \sigma(x w_2^{(2)}) \qquad \text{for } w_1^{(2)}, w_2^{(2)} \in \mathbb{R}.$$

To avoid transpose notation, we used $\mathbf{x} \in \mathbb{R}^{1\times d}$ to give one row of the data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, and row vector $\mathbf{h} \in \mathbb{R}^{1\times k_1}$. We use the superscript notation to distinguish between the weights in the first and last layer. It may seem counter-intuitive why we label $\mathbf{w}^{(2)}$ for the input layer, and $\mathbf{w}^{(1)}$ for the output layer, but you will see below it makes notation simpler to start indexing from the output layer.

Once we have $\mathbf{h}$, we can pretend that $\mathbf{h}$ is the new input representation and go ahead and learn a (generalized) linear model on this last layer. Let's consider two cases: $y \in \mathbb{R}$

and $y \in \{0, 1\}$. If $y \in \mathbb{R}$, we use linear regression for this last layer and so learn weights $\mathbf{w}^{(2)} \in \mathbb{R}^2$ such that $\mathbf{h}\mathbf{w}^{(2)}$ approximates the true output y. If $y \in \{0, 1\}$, we use logistic regression for this last layer and so learn weights $\mathbf{w}^{(2)} \in \mathbb{R}^2$ such that $\sigma(\mathbf{h}\mathbf{w}^{(2)})$ approximates the true output y. □

Now we consider the more general case with any d, k_1, m. To provide some intuition for this more general setting, we will begin with one hidden layer, for the sigmoid transfer function and cross-entropy output loss. For logistic regression we estimated $\mathbf{W} \in \mathbb{R}^{d \times m}$, with $f(\mathbf{x}\mathbf{W}) = \sigma(\mathbf{x}\mathbf{W}) \approx \mathbf{y}$. We will predict an output vector $\mathbf{y} \in \mathbb{R}^m$, because it will make later generalizations more clear-cut and make notation for the weights in each layer more uniform. When we add a hidden layer, we have two parameter matrices $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times k_1}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{k_1 \times m}$, where k_1 is the dimension of the hidden layer

$$\mathbf{h} = \sigma(\mathbf{x}\mathbf{W}^{(2)}) = \begin{bmatrix} \sigma(\mathbf{x}\mathbf{W}^{(2)}_{:1}) \\ \sigma(\mathbf{x}\mathbf{W}^{(2)}_{:2}) \\ \vdots \\ \sigma(\mathbf{x}\mathbf{W}^{(2)}_{:k_1}) \end{bmatrix} \in \mathbb{R}^{k_1}$$

where the sigmoid function is applied to each entry in $\mathbf{x}\mathbf{W}^{(2)}$ and $\mathbf{h}\mathbf{W}^{(1)}$. This hidden layer is the new set of features and again you will do the regular logistic regression optimization to learn weights on $\mathbf{h}$:

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{h}\mathbf{W}^{(1)}) = \sigma(\sigma(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)}).$$

With the probabilistic model and parameter specified, we now need to derive an algorithm to obtain those parameters. As before, we take a maximum likelihood approach and derive gradient descent updates. This composition of transfers seems to complicate matters, but we can still take the gradient w.r.t. our parameters. We simply have more parameters now: $\mathbf{W}^{(2)} \in \mathbb{R}^{k_1 \times d}, \mathbf{W}^{(1)} \in \mathbb{R}^{1 \times k_1}$. Once we have the gradient w.r.t. each parameter matrix, we simply take a step in the direction of the negative of the gradient, as usual. The gradients for these parameters share information; for computational efficiency, the gradient is computed first for $\mathbf{W}^{(1)}$, and duplicate gradient information sent back to compute the gradient for $\mathbf{W}^{(2)}$. This algorithm is typically called *back propagation*, which we describe next.

In general, we can compute the gradient for any number of hidden layers. Denote each differentiable transfer function $f_1, \ldots, f_H$, ordered with f_1 as the output transfer, and $k_1, \ldots, k_{H-1}$ as the hidden dimensions with $H - 1$ hidden layers. Then the output from the neural network is

$$f_1\left(f_2\left(\ldots f_{H-1}\left(f_H\left(\mathbf{x}\mathbf{W}^{(H)}\right)\mathbf{W}^{(H-1)}\right)\ldots\right)\mathbf{W}^{(1)}\right)$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{k_1 \times m}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{k_2 \times k_1}$, $\ldots, \mathbf{W}^{(H)} \in \mathbb{R}^{d \times k_{H-1}}$.

Backpropagation algorithm

We will start by deriving back propagation for two layers; the extension to multiple layers will be more clear given this derivation. Due to the size of the network, we will often learn

with stochastic gradient descent. Therefore, we will first compute this gradient assuming we only have one sample $(\mathbf{x}, \mathbf{y})$.

The back-propagation algorithm is simply gradient descent on a non-convex objective, with a careful ordering of computation to avoid repeating computation. In particular, one first propagates forward and computes variable $\mathbf{h} = f_2(\mathbf{x}\mathbf{W}^{(2)}) \in \mathbb{R}^{1\times k}$ and then $\hat{\mathbf{y}} = f_1(f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)}) = f_1(\mathbf{h}\mathbf{W}^{(1)})$. We then compute the error between our prediction $\hat{\mathbf{y}}$ and the true label. We take the gradient of this error (loss) w.r.t. to our parameters; in this case, for efficient computation, the best ordering is to compute the gradient w.r.t. to the last parameter $\mathbf{W}^{(1)}$ first, and then $\mathbf{W}^{(2)}$. This is the reason for the term back-propagation, since the error is propagated backward from the last layer first.

The choices then involve picking the transfers at each layer, the number of hidden nodes and the loss for the last layer. The matching convex loss $L(\cdot, y)$ depends on the chosen $p(y|\mathbf{x})$ and corresponding transfer function for the last layer of the neural network, just as with generalized linear models. For ease of notation, we define this error function as

$$\text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) = \sum_{k=1}^{m} L(f_1(f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)}_{:k}), \mathbf{y}_k)$$

for one sample $(\mathbf{x}, \mathbf{y})$. For example, for $p(y = 1|\mathbf{x})$ Gaussian and identity transfer f_2, we get $\text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) = (f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)} - y)^2$. If $p(y = 1|\mathbf{x})$ is a Bernoulli distribution, then we would chose the logistic regression loss (the cross entropy).

As before, we will compute gradients of the loss w.r.t. our parameters. First, we take the partial derivative w.r.t. the parameters $\mathbf{W}^{(1)}$ (assuming $\mathbf{W}^{(2)}$ is fixed).

$$\begin{aligned}
\frac{\partial \text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial \mathbf{W}^{(1)}_{jk}} &= \frac{\partial L(f_1(f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)}), \mathbf{y})}{\partial \mathbf{W}^{(1)}_{jk}} \\
&= \left(\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k}\right) \frac{\partial \hat{\mathbf{y}}_k}{\partial \mathbf{W}^{(1)}_{jk}} \qquad \triangleright\ \hat{\mathbf{y}}_k = f_1(\mathbf{h}\mathbf{W}^{(1)}_{:k})
\end{aligned}$$

where only $\hat{\mathbf{y}}_k$ is affected by $\mathbf{W}^{(1)}_{jk}$ in the loss, and so the gradient for the others is zero. Continuing,

$$\begin{aligned}
\frac{\partial \text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial \mathbf{W}^{(1)}_{jk}} &= \left(\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k}\right) \frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} \frac{\partial \boldsymbol{\theta}^{(1)}_k}{\partial \mathbf{W}^{(1)}_{jk}} \qquad \triangleright\ \boldsymbol{\theta}^{(1)}_k = \mathbf{h}\mathbf{W}^{(1)}_{:k} \\
&= \left(\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k}\right) \frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} \mathbf{h}_j
\end{aligned}$$

At this point these equations are abstract; but they are simple to compute for the losses and transfers we have examined. For example, for $L(\hat{\mathbf{y}}_k, \mathbf{y}_k) = \frac{1}{2}(\hat{\mathbf{y}}_k - \mathbf{y}_k)^2$, and f_2 the identity, we get

$$\begin{aligned}
\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} &= (\hat{\mathbf{y}}_k - \mathbf{y}_k) \\
\frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} &= 1
\end{aligned}$$

giving

$$\frac{\partial \text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial \mathbf{W}^{(1)}_{jk}} = \left(\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k}\right) \frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} \mathbf{h}_j = (\hat{\mathbf{y}}_k - \mathbf{y}_k)\mathbf{h}_j.$$

The gradient update is as usual with $\mathbf{W}^{(1)} = \mathbf{W}^{(1)} - \alpha(\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^\top$ for some step-size α.

Next, we compute the partial gradient with respect to $\mathbf{W}^{(2)}$. Now, however, the entire output variable $\mathbf{y} \in \mathbb{R}^{1\times m}$ is affected by the choice of $\mathbf{W}^{(2)}_{ij}$ for all $i \in \{1, \ldots, k_2\}$, $j \in \{1, \ldots, k_1\}$. Therefore, we need to take the partial derivative w.r.t. all of $\mathbf{y}$.

$$\begin{aligned}
\frac{\partial \text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial \mathbf{W}^{(2)}_{ij}} &= \frac{\partial \sum_{k=1}^m L(f_1(f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)}_{:k}), \mathbf{y}_k)}{\partial \mathbf{W}^{(2)}_{ij}} \\
&= \sum_{k=1}^m \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial \hat{\mathbf{y}}_k}{\partial \mathbf{W}^{(2)}_{ij}} && \triangleright\ \hat{\mathbf{y}}_k = f_1(\mathbf{h}\mathbf{W}^{(1)}_{:k}) = f_1(\boldsymbol{\theta}^{(1)}_k) \\
&= \sum_{k=1}^m \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} \frac{\partial \boldsymbol{\theta}^{(1)}_k}{\partial \mathbf{W}^{(2)}_{ij}}.
\end{aligned}$$

Continuing,

$$\begin{aligned}
\frac{\partial \boldsymbol{\theta}^{(1)}_k}{\partial \mathbf{W}^{(2)}_{ij}} = \frac{\partial \mathbf{h}\mathbf{W}^{(1)}_{:k}}{\partial \mathbf{W}^{(2)}_{ij}} &= \frac{\partial \sum_{l=1}^k \mathbf{h}_l \mathbf{W}^{(1)}_{lk}}{\partial \mathbf{W}^{(2)}_{ij}} \\
&= \frac{\partial \sum_{l=1}^k f_2(\mathbf{x}\mathbf{W}^{(2)}_{:l})\mathbf{W}^{(1)}_{lk}}{\partial \mathbf{W}^{(2)}_{ij}} \\
&= \sum_{l=1}^k \mathbf{W}^{(1)}_{lk} \frac{\partial f_2(\mathbf{x}\mathbf{W}^{(2)}_{:l})}{\partial \mathbf{W}^{(2)}_{ij}} \\
&= \mathbf{W}^{(1)}_{jk} \frac{\partial f_2(\mathbf{x}\mathbf{W}^{(2)}_{:j})}{\partial \mathbf{W}^{(2)}_{ij}}
\end{aligned}$$

because $\frac{\partial f_2(\mathbf{x}\mathbf{W}^{(2)}_{:l})}{\partial \mathbf{W}^{(2)}_{ij}} = 0$ for $l \neq j$. Now continuing the chain rule

$$\begin{aligned}
\frac{\partial f_2(\mathbf{x}\mathbf{W}^{(2)}_{:j})}{\partial \mathbf{W}^{(2)}_{ij}} &= \frac{\partial f_2(\boldsymbol{\theta}^{(2)}_j)}{\partial \boldsymbol{\theta}^{(2)}_j} \frac{\partial \boldsymbol{\theta}^{(2)}_j}{\partial \mathbf{W}^{(2)}_{ij}} && \triangleright\ \boldsymbol{\theta}^{(2)}_j = \mathbf{x}\mathbf{W}^{(2)}_{:j} \\
&= \frac{\partial f_2(\boldsymbol{\theta}^{(2)}_j)}{\partial \boldsymbol{\theta}^{(2)}_j} \mathbf{x}_i.
\end{aligned}$$

Putting this back together, we get

$$\begin{aligned}
\frac{\partial \text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial \mathbf{W}^{(2)}_{ij}} &= \sum_{k=1}^m \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} \frac{\partial \boldsymbol{\theta}^{(1)}_k}{\partial \mathbf{W}^{(2)}_{ij}} \\
&= \sum_{k=1}^m \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_1(\boldsymbol{\theta}^{(1)}_k)}{\partial \boldsymbol{\theta}^{(1)}_k} \mathbf{W}^{(1)}_{jk} \frac{\partial f_2(\boldsymbol{\theta}^{(2)}_j)}{\partial \boldsymbol{\theta}^{(2)}_j} \mathbf{x}_i.
\end{aligned}$$

Notice that some of gradient is the same as for $\mathbf{W}^{(1)}$, i.e.

$$\boldsymbol{\delta}_k^{(1)} = \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_1(\boldsymbol{\theta}_k^{(1)})}{\partial \boldsymbol{\theta}_k^{(1)}}$$

Computing these components only needs to be done once for $\mathbf{W}^{(1)}$, and this information propagated back to get the gradient for $\mathbf{W}^{(2)}$. The difference is in the gradient $\frac{\partial \boldsymbol{\theta}^{(1)}}{\partial \mathbf{W}^{(2)}}$, because $\mathbf{h}$ relies on $\mathbf{W}^{(2)}$. For $\mathbf{W}^{(1)}$, $\mathbf{h} = f_2(\mathbf{x}_i \mathbf{W}^{(2)})$ is a constant, and so does not affect the gradient for $\mathbf{W}^{(1)}$. The final gradient is

$$\begin{aligned}
\frac{\partial \text{Err}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)})}{\partial \mathbf{W}_{ij}^{(2)}} &= \left(\sum_{k=1}^{m} \boldsymbol{\delta}_k^{(1)} \mathbf{W}_{jk}^{(1)} \right) \frac{\partial f_2(\boldsymbol{\theta}_j^{(2)})}{\partial \boldsymbol{\theta}_j^{(2)}} \mathbf{x}_i \\
&= \left(\mathbf{W}_{j:}^{(1)} \boldsymbol{\delta}^{(1)} \right) \frac{\partial f_2(\boldsymbol{\theta}_j^{(2)})}{\partial \boldsymbol{\theta}_j^{(2)}} \mathbf{x}_i
\end{aligned}$$

If another layer is added before $\mathbf{W}^{(2)}$, then the information propagated backward is

$$\boldsymbol{\delta}_j^{(2)} = \left(\mathbf{W}_{j:}^{(1)} \boldsymbol{\delta}^{(1)} \right) \frac{\partial f_2(\boldsymbol{\theta}_j^{(2)})}{\partial \boldsymbol{\theta}_j^{(2)}}$$

and $\mathbf{x}_i$ is replaced with $\mathbf{h}_i^{(2)}$. The gradient for $\mathbf{W}_{ij}^{(3)}$ is

$$\left(\mathbf{W}_{j:}^{(2)} \boldsymbol{\delta}^{(2)} \right) \frac{\partial f_3(\boldsymbol{\theta}_j^{(3)})}{\partial \boldsymbol{\theta}_j^{(3)}} \mathbf{x}_i$$

Example 19: Let $p(y = 1|\mathbf{x})$ be a Bernoulli distribution, with f_1 and f_2 both sigmoid functions. The loss is the cross-entropy. We can derive the two-layer update rule with these settings, by plugging-in above.

$$\begin{aligned}
L(\hat{y}, y) &= -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) && \triangleright \text{ cross-entropy} \\
\frac{\partial L(\hat{y}, y)}{\partial \hat{y}} &= -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \\
f_2(\mathbf{x}\mathbf{W}_{:j}^{(2)}) &= \sigma(\mathbf{x}\mathbf{W}_{:j}^{(2)}) = \frac{1}{1 + \exp(-\mathbf{x}\mathbf{W}_{:j}^{(2)})} \\
f_1(\mathbf{h}\mathbf{W}_{:k}^{(1)}) &= \sigma(\mathbf{h}\mathbf{W}_{:k}^{(1)}) = \frac{1}{1 + \exp(-\mathbf{h}\mathbf{W}_{:k}^{(1)})} \\
\partial \sigma(\theta) &= \sigma(\theta)(1 - \sigma(\theta))
\end{aligned}$$

Now we can compute the backpropagation update by first propagating forward

$$\begin{aligned}
\mathbf{h} &= \sigma(\mathbf{x}\mathbf{W}^{(2)}) \\
\hat{\mathbf{y}} &= \sigma(\mathbf{h}\mathbf{W}^{(1)})
\end{aligned}$$

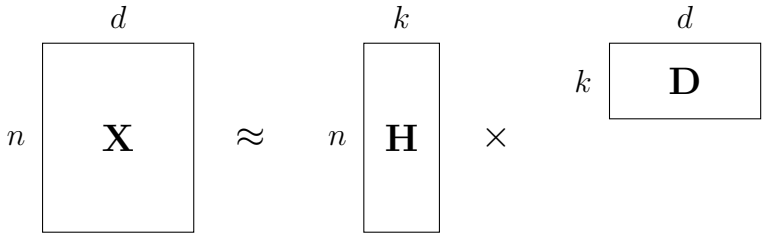


Figure 9.4: Matrix factorization of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$.

and then propagating the gradient back

$$\begin{aligned}
\boldsymbol{\delta}_k^{(1)} &= \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_1(\boldsymbol{\theta}_k^{(1)})}{\partial \boldsymbol{\theta}_k^{(1)}} \\
&= \left(-\frac{\mathbf{y}_k}{\hat{\mathbf{y}}_k} + \frac{1 - \mathbf{y}_k}{1 - \hat{\mathbf{y}}_k} \right) \hat{\mathbf{y}}_k (1 - \hat{\mathbf{y}}_k) \\
&= -\mathbf{y}_k (1 - \hat{\mathbf{y}}_k) + (1 - \mathbf{y}_k) \hat{\mathbf{y}}_k \\
&= \hat{\mathbf{y}}_k - \mathbf{y}_k \\
\frac{\partial}{\partial \mathbf{W}_{jk}^{(1)}} &= \boldsymbol{\delta}_k^{(1)} \mathbf{h}_j \\
\boldsymbol{\delta}_j^{(2)} &= \left(\mathbf{W}_{j:}^{(1)} \boldsymbol{\delta}^{(1)} \right) \mathbf{h}_j (1 - \mathbf{h}_j) \\
\frac{\partial}{\partial \mathbf{W}_{ij}^{(2)}} &= \boldsymbol{\delta}_j^{(2)} \mathbf{x}_i
\end{aligned}$$

The update simply consists of stepping in the direction of these gradients, as is usual for gradient descent. We start with some initial $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ (say filled with random values), and then apply the gradient descent rules with these gradients. □

9.2.2 Unsupervised learning and matrix factorization

Another strategy to obtaining a new representation is through matrix factorization. The data matrix $\mathbf{X}$ is factorized into a dictionary $\mathbf{D}$ and a basis or new representation $\mathbf{H}$ (see Figure 9.4). In fact, many unsupervised learning algorithms (e.g., dimensionality reduction, sparse coding) and semi-supervised learning algorithms (e.g., supervised dictionary learning) can actually be formulated as matrix factorizations. We will look at k-means clustering and principal components analysis as an example. The remaining algorithms are simply summarized in the below table. This general approach to obtaining a new representation using factorization is called **dictionary learning**.

K-means clustering is an unsupervised learning problem to group data points into k clusters by minimizing distances to the mean of each cluster. This problem is not usually thought of as a representation learning approach, because the cluster number is not typically used as a representation. However, we nonetheless start with k-means because it is an intuitive example of how these unsupervised learning algorithms can be thought of as matrix

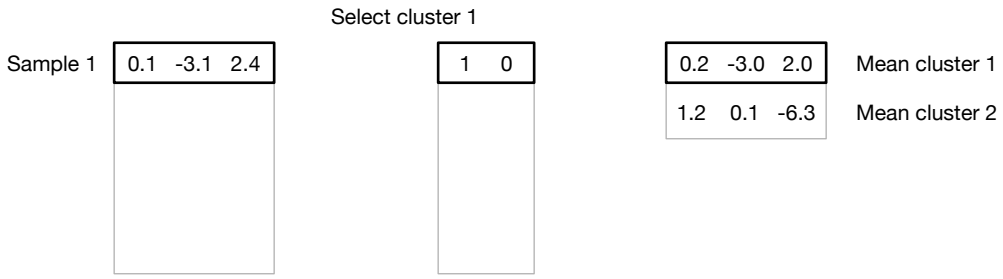


Figure 9.5: K-means clustering as a matrix factorization for data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$.

factorization. Further, the clustering approach can be seen as a representation learning approach, because it is a learned discretization of the space. We will discuss this view of k-means after discussing it as a matrix factorization.

Imagine that you have two clusters ($k = 2$), with data dimension $d = 3$. Let $\mathbf{d}_1$ be the mean for cluster 1 and $\mathbf{d}_2$ the mean for cluster 2. The goal is to minimize the squared ℓ_2 distance of each data point $\mathbf{x}$ to its cluster center

$$\|\mathbf{x} - \sum_{i=1}^{2} 1\,(\mathbf{x} \text{ in cluster } i)\,\mathbf{d}_i\|_2^2 = \|\mathbf{x} - \mathbf{h}\mathbf{D}\|_2^2$$

where $\mathbf{h} = [1 \quad 0]$ or $\mathbf{h} = [0 \quad 1]$ and $\mathbf{D} = [\mathbf{d}_1 \; ; \; \mathbf{d}_2]$. An example is depicted in Figure 9.5. For a point $\mathbf{x} = [0.1 \quad -3.1 \quad 2.4]$, $\mathbf{h} = [1 \quad 0]$, meaning it is placed in cluster 1 with mean $\mathbf{d}_1 = [0.2 \quad -3.0 \quad 2.0]$. It would incur more error to place $\mathbf{x}$ in cluster 2 which has a mean that is more dissimilar: $\mathbf{d}_2 = [1.2 \quad 0.1 \quad -6.3]$.

The overall minimization is defined across all the samples, giving loss

$$\min_{\substack{\mathbf{H} \in \{0,1\}^{n \times k}, \mathbf{1}\mathbf{H} = \mathbf{1} \\ \mathbf{D} \in \mathbb{R}^{k \times d}}} \|\mathbf{X} - \mathbf{H}\mathbf{D}\|_F^2.$$

Different clusters vectors $\mathbf{h}$ are learned for each $\mathbf{x}$, but the dictionary of means is shared amongst all the data points. The specified optimization should pick dictionary $\mathbf{D}$ of means that provides the smallest distances to points in the training dataset.

Principal components analysis (PCA) is a standard dimensionality reduction technique, where the input data $\mathbf{x} \in \mathbb{R}^{1 \times d}$ is projected into a lower dimensional $\mathbf{h} \in \mathbb{R}^{1 \times k}$ spanned by the space of principal components. These principal components are the directions of maximal variance in the data. To obtain these k principal components $\mathbf{D} \in \mathbb{R}^{k \times d}$, the common solution technique is to obtain the singular value decomposition of the data matrix $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \in \mathbb{R}^{n \times d}$, giving

$$\mathbf{D} = \mathbf{V}_k^\top \in \mathbb{R}^{k \times d}$$
$$\mathbf{H} = \mathbf{U}_k \mathbf{\Sigma}_k \in \mathbb{R}^{n \times k}$$

where $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ consists of the top largest k singular values (in descending order) and $\mathbf{U}_k \in \mathbb{R}^{n \times k}$ and $\mathbf{V}_k \in \mathbb{R}^{k \times d}$ are the corresponding singular vectors, i.e., $\mathbf{U}_k = \mathbf{U}_{:,1:k}$ and $\mathbf{V}_k = \mathbf{V}_{:,1:k}$. The new representation for $\mathbf{X}$ (using PCA) is this $\mathbf{H}$. Note that PCA does

not subselect features, but rather creates new features: The generated $\mathbf{h}$ is not a subset of the original $\mathbf{x}$.

This dimensionality reduction technique can also be formulated as a matrix factorization. The corresponding optimization has been shown to be

$$\min_{\mathbf{D}\in\mathbb{R}^{k\times d},\mathbf{H}\in\mathbb{R}^{n\times k}} \|\mathbf{X}-\mathbf{H}\mathbf{D}\|_F^2$$

One simple way to see why is to recall the well-known Eckart-Young-Mirsky theorem that the rank k matrix $\hat{\mathbf{X}}$ that best approximates $\mathbf{X}$, in terms of minimal Frobenius norm, is $\hat{\mathbf{X}} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$.

As with k-means clustering, it may be hard to immediately see why $\mathbf{h}$ generated by PCA could be useful as a representation. In fact, PCA is often used for visualization, and so is not always used for representation learning. For visualization, the projection is often aggressive to two or three dimensions. In general, however, the projection to lower dimensions has the property that it removes noise and maintains only the most meaningful directions. This projection, therefore, helps speed learning by reducing the number of features and promoting generalization, by preventing overfitting to the noise.

Sparse coding takes a different approach, where the input data is expanded into a sparse representation. Sparse coding is biologically motivated [15], based on sparse activations for memory in the mammalian brain. Another interpretation is that sparse coding effectively discretizes the space, like k-means clustering, but with overlapping clusters and an associated magnitude of how much a point belongs to that cluster.

A common strategy to obtain sparse representations is to use a sparse regularizer on the learned representation $\mathbf{h}$. This corresponds to the optimization

$$\min_{\mathbf{D}\in\mathbb{R}^{k\times d},\mathbf{H}\in\mathbb{R}^{n\times k}} \|\mathbf{X}-\mathbf{H}\mathbf{D}\|_F^2 + \lambda\sum_{i=1}^{k}\|\mathbf{H}_{:i}\|_1 + \lambda\sum_{i=1}^{k}\|\mathbf{D}_{i:}\|_2^2$$

As discussed in Section 5.4.2, the ℓ_1 regularizer promotes zeroed entries, and so prefers $\mathbf{H}$ with as many zeros as possible. A regularizer is also added to $\mathbf{D}$, to ensure that $\mathbf{D}$ does not become too large; otherwise, all the weight in $\mathbf{D}\mathbf{H}$ would be shifted to $\mathbf{D}$. As an exercise, see if you can explain why, and what this means for identifiability.

In general, there are many variants of unsupervised learning algorithms that actually correspond to factorizing the data matrix; additional details are given in Appendix D. We additionally give details on how to learn these factorizations in the appendix. As with previous algorithms, they are simply gradient descent on the (matrix) variables. The only distinction here is that it is common to use block coordinate descent, instead of the more standard gradient descent algorithm. This distinction is minor, and it would be perfectly valid to use standard gradient descent.

Chapter 10

Evaluation of Learning Algorithms

The majority of this book has focused on algorithm derivation and obtaining models, but we have yet to address how to evaluate these models. The maximum likelihood formalism for deriving learning algorithms provides some consistency results, where in the limit of samples we can discuss the convergence point of an estimator. In practice, however, we would like to evaluate the algorithms based on a finite sample. Imagine a setting where you learn two models, say using logistic regression with two different regularization parameters. Which of these two models is "better"? What does it even mean to say better? Do you want to say the model is better for this problem (data setting), or across multiple problems? Are we trying to compare algorithms or models obtained from a specific instance of an algorithm? How can we be confident that the measured performance accurately reflects the performance we expect to see on new data? These questions are largely separate from our previous questions of effectively optimizing a specified objective, and rather starts to ask questions about the properties of that objective and about empirical properties of learned models.

In this chapter, we provide theoretical and empirical tools to better evaluate the properties of learning algorithms. We begin with some basic finite-sample theoretical results, that relate the complexity of the model class to the number of samples required to obtain a reasonable estimate of expected error (generalization error). This section will also introduce the ideas of optimizing over a function class, and our goals for obtaining the best model in terms of generalization error. The area dealing with these types of theoretical characterizations is called *statistical learning theory*. We will discuss one result using *concentration inequalities* and *Rademacher complexity* to characterize model-class complexity; for further information, you could consider this tutorial on the topic [7].

Then, we will discuss how to compare algorithms empirically. In most real-world settings, you will choose between algorithms based on their performance on available data. You want this choice to be reflective of how well those algorithms will perform on new data. Towards this goal, we will discuss how to split data and how to use statistical significance tests to provide some level of confidence that one algorithm or model is better than another, under some specific criteria. We will rarely be able to make strong conclusions based on experiments, but we can build up some evidence on the algorithm properties.

These tools are arguably the most critical aspects of properly using machine learning algorithms in practice. One can learn a complex model, but without any understanding of how it is expected to perform in practice on new data, it is not viable to actual use these models. Whether an algorithm is used for scientific purposes or deployed in real systems, have an understanding of its properties both theoretically and empirically is key to obtain expected outcomes. This chapter only begins to scratch the surface of these tools, with the goal to pique your interest and direct you towards more material for learning about

evaluation.

10.1 A brief introduction to generalization bounds

Our goal throughout this book has been to obtain a function, based on a set of examples, that predicts accurately: produces low expected error across the space of possible examples. We cannot, however, measure the expected error. Statistically, we know that with a sufficient sample, we can approximate an expectation. Here, we quantify this more carefully for learned functions.

Our goal more precisely is to select a function from a function class $\mathcal{H}$ to minimize a loss function $\ell : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ in expectation over all pairs $(\mathbf{x}, y)$

$$\min_{f \in \mathcal{H}} \mathbb{E}[\ell(f(\boldsymbol{X}), Y)].$$

For example, in linear regression, $\mathcal{H} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for any } \mathbf{w} \in \mathbb{R}^d\}$. This space of functions $\mathcal{H}$ represents all possible linear functions of inputs $\mathbf{x} \in \mathbb{R}^d$, to produce a scalar output. Our goal in linear regression was to minimize a proxy to the true expected error, i.e., the sample error: $\frac{1}{n}\sum_{i=1}^n \ell(f(\mathbf{x}_i), y_i)$. Now a natural question to ask is: does this sample error provide an accurate estimate of the true expected error? And what does it tell us about the true generalization performance, i.e., true expected error?

Let's start with a simple example, using linear regression. Assume a bounded function class $\mathcal{H}$, where $\mathcal{H} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for any } \mathbf{w} \in \mathbb{R}^d \text{ such that } \|\mathbf{w}\|_2 \leq B_w\}$ for some finite scalar $B_w > 0$. Assume the input features come from a bounded space, such that for all $\mathbf{x}$, $\|\mathbf{x}\|_2 \leq B_x$ for some finite scalar $B_x > 0$, and further that the outputs $y \in [-B_y, B_y]$ for some $B_y > 0$. Assume we use loss $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$, which is (locally) Lipschitz continuous for our bounded region, with Lipschitz constant $c = B_y + B_x B_w$. This is because $|\hat{y}| \leq B_x B_w$ and

$$\left|\frac{d\ell(\hat{y}, y)}{d\hat{y}}\right| = |\hat{y} - y| \leq |\hat{y}| + |y| \leq B_y + B_x B_w.$$

Further, because $y \in [-B_y, B_y]$, we know the loss is bounded as

$$\ell(\hat{y}, y) = \tfrac{1}{2}(\hat{y} - y)^2 \leq \tfrac{1}{2}(B_y^2 + B_x^2 B_w^2).$$

For approximate error

$$\widehat{\mathrm{Err}}(f) = \frac{1}{n}\sum_{i=1}^n \ell(f(\mathbf{x}_i), y_i)$$

and true error

$$\mathrm{Err}(f) = \mathbb{E}[\ell(f(\boldsymbol{X}), Y)] = \int_{\mathcal{X} \times \mathcal{Y}} p(\mathbf{x}, y)\ell(f(\mathbf{x}), y) d\mathbf{x} dy$$

using Equation 10.2 below, we get that with probability $1 - \delta$, for $\delta \in (0, 1]$,

$$\mathrm{Err}(f) \leq \widehat{\mathrm{Err}}(f) + \frac{2cB_xB_w}{\sqrt{n}} + \tfrac{1}{2}(B_y^2 + B_x^2 B_w^2)\sqrt{\frac{\ln(1/\delta)}{2n}}. \tag{10.1}$$

With increasing samples n, the second two terms disappear and the sample error approaches the true expected error. This bound show the rate at which this discrepancy disappears. For a higher confidence—small δ making $\ln(1/\delta)$ larger—more samples are need for the third term to be small. This third term is obtained using *concentration inequalities*, which enable us to state the rate at which a sample mean gets close to its expected value. For possibly large values of features or learned weights, the second term can be big and can again require the more samples. The second term reflects the properties of our function class: a simpler class, with small bounded weights, can have a more accurate estimate of the loss on a smaller number of samples. More generally, this complexity measure is called the *Rademacher complexity.*[1] For the linear functions above, with bounded ℓ_2 norms for $\mathbf{x}$, $\mathbf{w}$, the Rademacher complexity is bounded as $R_n(\mathcal{H}) \leq B_x B_y/\sqrt{n}$ (see [10, Equation 3]).

In the next few sections, we provide a generalization result for more general functions, as well as required background to determine that result.

10.1.1 Concentration inequalities

We will examine the use of concentration inequalities with one common example: Hoeffding's inequality. For the generalization bound below, a generalization is used, called McDiarmid's inequality.

For i.i.d. random variables $X_1, \ldots, X_n$, such that $0 \leq X_i \leq 1$, let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ be the sample average. Then Hoeffding's inequality states that for any ϵ

$$\Pr(\bar{X} - \mathbb{E}[\bar{X}] \geq \epsilon) \leq \exp(-2n\epsilon^2).$$

We start by setting this probability value to δ, so that we can say with probability δ, $\Pr(\bar{X} - \mathbb{E}[\bar{X}] \geq \text{function}(\delta))$. We can solve for ϵ in terms of δ, to get

$$\delta = \exp(-2n\epsilon^2) \qquad \implies \epsilon = \pm\sqrt{\frac{\ln(1/\delta)}{2n}}.$$

We can either set ϵ to $\sqrt{\frac{\ln(1/\delta)}{2n}}$ or $-\sqrt{\frac{\ln(1/\delta)}{2n}}$, to bound $\bar{X}$ to be near $\mathbb{E}[\bar{X}]$ from both above and below. We get that with probability $1-\delta$, $\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \leq |\epsilon| = \sqrt{\frac{\ln(1/\delta)}{2n}}$.

This concentration inequality makes few assumptions about the random variables, and does not require any distributional assumptions. Consequently, the rate of convergence to the true mean is only $1/\sqrt{n}$. Faster rates can be obtained with more assumptions.

10.1.2 Complexity of a function class

Rademacher complexity of a function class characterizes the overfitting ability of functions, on a particular sample. Function class that are typically more complex are more likely to be able to fit random noise, and so have higher Rademacher complexity. The empirical Rademacher complexity, for a sample $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ —where typically we consider $\mathbf{z}_i = (\mathbf{x}_i, y_i)$

[1] If you have heard of VC dimension, we will discuss the connection between Rademacher and VC dimension below. They both play a role in identifying the complexity of a function class.

— is defined as[2]

$$\hat{R}_n(\mathcal{H}) = \mathbb{E}\left[\max_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right]$$

where the expectation is over i.i.d. random variables $\sigma_1, \ldots, \sigma_n$ chosen uniformly from $\{-1, 1\}$. This choice reflects how well the function class can correlate with this random noise. Consider for example if $f(\mathbf{x})$ predicts 1 or -1, as in binary classification. If there exists a function in the class of functions that can perfectly match the sign of the randomly sampled σ_i, then that function produces the highest value $\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)$. The empirical Rademacher complexity for a function class is high, if for any randomly sampled σ_i, there exists such a function within the function class (can be a different function for each $\sigma_1, \ldots, \sigma_n$). The Rademacher complexity is the expected empirical Rademacher complexity, over all possibly samples of n instances.

For function classes with high Rademacher complexity, error on the training set is unlikely to be reflective of the generalization error, until there is a sufficient number of samples. This is reflected in the generalization bound in Section 10.1.3.

Connection to VC dimension: The complexity of a function class can also be characterized by the VC dimension. The idea of VC dimension to characterize the number of points that can be separated (or shattered) by a function class. Simple functions have low VC dimension, because they are not complex enough to separate many points. More complex functions, that enable complex boundaries, have higher VC dimension. For example, for functions of the form $f((x_1, x_2)) = \text{sign}(x_1 w_1 + x_2 w_2 + w_0)$, the VC dimension is 3; more generally, for $\mathbf{x} \in \mathbb{R}^d$, the VC dimension is $d + 1$. VC dimension is a similar idea to Rademacher complexity, but it is restricted to binary classifiers. For this reason, we directly discuss the Rademacher complexity, which for binary classifiers can be bounded in terms of the VC dimension. By Sauer's Lemma, we can typically bound the Rademacher complexity of a hypothesis class by $\sqrt{\frac{2\text{VC-dimension} \ln n}{n}}$.

10.1.3 Generalization bounds

The generalization bound for a class of models can be obtained by combining the concentration inequalities to bound deviation from the mean for fewer samples, and using the Rademacher complexity to bound the difference between the sample error and true expected error across all functions in the function class. We additionally need to restrict the set of losses. We assume that the losses are Lipschitz with constant c, meaning that they do not change too quickly in a region, with c indicating the rate of change. Further, we also assume that the loss is bounded by b, i.e., attains values in $[-b, b]$. As above, if $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ is i.i.d., then with probability $1 - \delta$, for every $f \in \mathcal{H}$,

$$\mathbb{E}[\ell(f(\boldsymbol{X}), Y)] \leq \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) + 2cR_n(\mathcal{H}) + b\sqrt{\frac{\ln(1/\delta)}{2n}} \tag{10.2}$$

For a more precise theorem statement and a proof, see [3, Theorem 7] and [10, Theorem 1].

[2]Here we are being a bit loose and using maximum instead of supremum, to avoid burdening the reader with new terminology. We usually deal with function classes $\mathcal{H}$ where using the supremum is equivalent to using the maximum. The supremum is used when a set does not contain a maximal point (e.g., $[0, 1)$), where the supremum provides the closest upper bound (e.g., 1 for $[0, 1)$).

10.2 Comparison of Learning Algorithms

To empirically evaluate algorithms, we can consider a setting with one or more algorithms on one or more datasets. Depending on the setting, different evaluations will be employed. For a nice overview of evaluation for machine learning algorithms, see [9].

For now, let's start with a simple case, where we compare two algorithms and use the binomial test. Suppose we have a set of learning problems $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_m$ and wish to compare learning algorithms a_1 and a_2. We can carry out such a comparison using a counting test as follows: for each data set both algorithms are evaluated in terms of the chosen performance measure and the algorithm with a higher performance accuracy is awarded a win, while the other one is given a loss (in case of exactly the same performance, we can provide a win/loss randomly).

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$		$\mathcal{D}_{m-1}$	$\mathcal{D}_m$
a_1	1	0	1	1	$\cdots$	0	1
a_2	0	1	0	0		1	0

Table 10.1: A counting test where learning algorithms a_1 and a_2 are compared on a set of m independent data sets. An algorithm with a better performance on a particular data set collects a win (1), whereas the other algorithm collects a loss (0).

We are now interested in providing statistical evidence that say algorithm a_1 is better than algorithm a_2. Suppose a_1 has k wins out of m and algorithm a_2 has $m - k$ wins, as shown in Table 10.1. We would like to evaluate the null hypothesis H_0 that algorithms a_1 and a_2 have the same performance by providing an alternative hypothesis H_1 that algorithm a_1 is better than a_2. In short,

$$H_0\text{: quality}(a_1) = \text{quality}(a_2)$$
$$H_1\text{: quality}(a_1) > \text{quality}(a_2)$$

If the null hypothesis is true, the win/loss on each data set will be equally likely and determined by minor variation. Therefore, the probability of a win on any data set will be roughly equal to $p = 1/2$. Now, we can express the probability that algorithm a_1 collected k wins or more under the null hypothesis using binomial distribution

$$P = \sum_{i=k}^{m} \binom{m}{i} p^i (1-p)^{m-i}$$

and refer to it as the P-value. This value is the probability of k wins, plus the probability of $k+1$ wins, up to the probability of m wins, under the null hypothesis. A typical approach in these cases is to establish a significance value, say, $\alpha = 0.05$ and reject the null hypothesis if $P \leq \alpha$. If the P-value is greater than α we say that there is insufficient evidence for rejecting H_0. For sufficiently low P-values, we may conclude that there is sufficient evidence that algorithm a_1 is better than algorithm a_2.

The choice of the significance threshold α is somewhat arbitrary. Typically, 5% is a reasonable value, but lower values indicate that the particular situation of k wins out of m was so unlikely, that we can consider the evidence for rejecting H_0 very strong. Being

able to reject the null hypothesis provides some confidence that the result did not occur by chance.

More generally, we can consider other statistical significance tests based on the distributions of the performance measures. In the above example, a binomial distribution was appropriate. If instead we considered the actual errors on the datasets, then we have pairs of real values. In this case, a common choice is the paired t-test, if both errors appear to be distributed normally and if they have similar variance. The paired t-test takes in the sampled differences between the algorithms (line 3 in Table 10.2), $d_1, \ldots, d_m$. Because again our null hypothesis is that the algorithms perform equally, under the null hypothesis the mean of these differences is 0. If the differences are normally distributed, then for the sample average $\bar{d} = \frac{1}{m}\sum_{i=1}^{m} d_i$ and sample standard deviation $S_d = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(d_i - \bar{d})^2}$, the random variable $t = \frac{\bar{d}-0}{S_d/\sqrt{m}}$ is distributed according to the Student's t-distribution. The Student's t-distribution is approximately like a normal distribution, with a degrees-of-freedom parameter $m-1$ that makes the distribution look more like a normal distribution as m becomes larger.

We can now ask about the probability of this random variable T, relative to the computed statistic. If we only care about knowing if algorithm 1 is better than algorithm 2, we conduct a one-tailed test. If the probability that T is larger than t, i.e., $p = Pr(T > t)$, is small, then we obtain some evidence that algorithm 1 is better than algorithm 2. To test if algorithm 1 is better than algorithm 2, we can swap the order of the difference; if $p = Pr(T > -t)$ is small, then we obtain some evidence that algorithm 2 is better than algorithm 1. These are both one-tailed tests, reflecting the probabilities at one end of the tails of the distribution. A two-tailed test instead asks if the two algorithms are different; in this case, one would use $p = Pr(T > |t|)$.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$		$\mathcal{D}_{m-1}$	$\mathcal{D}_m$
a_1	0.11	0.08	0.15	0.12	$\cdots$	0.07	0.09
a_2	0.10	0.09	0.11	0.12	$\cdots$	0.10	0.09
d	0.01	-0.01	0.04	0.0	$\cdots$	-0.03	0.0

Table 10.2: A table of errors for two learning algorithms a_1 and a_2 are compared on a set of m independent data sets. The last row contains the differences, which are used for the paired t-test.

If the paired samples are not normally distributed, other tests are more suitable. Further, there are some tests that do not make distributional assumptions, and rather are non-parametric. For a summary of which test to use in different settings, see [9, Section 6.3]

10.3 Obtaining samples of error

A key step in comparing algorithms is to obtain valid measures of performance for the comparison. So far, we have assumed that these are given. One approach to obtain unbiased samples of the error is to keep a hold-out test set. Imagine m samples are set in reserve, on which the algorithms are not trained and which we cannot look at until we are ready to evaluate. We can train two models on the training set, and then obtain m paired samples

of error. We can then use the paired t-test to make claims about if the two models are statistically significantly different for the problem.

However, there are two key disadvantages to using a hold-out test set. First, usually we want to use all the data for training. Unless there is more data than can be used, keeping a hold-out test set is typically not practical. Even in this age of huge datasets, we still typically want to learn on as much (quality) data as possible. Second, once this hold-out test set has been used for evaluation, we cannot use it again because it will not provide an unbiased estimate of the expected error. For example, after getting performance of your models on that test set, one could go back and adjust meta-parameters such as the regularization parameters. However, once you have done this, the test-set has influenced the learned models and is likely to produce an optimistic estimate of performance on new data. Therefore, this hold-out test-set can only be used once.

An alternative approach to obtain estimates of error is to use resampling techniques from the whole dataset. Two common resampling techniques are k-fold cross-validation and bootstrap resampling. In the first, the data is partitioned into k disjoint sets (folds). The model is trained on $k-1$ of the folds, and tested on the other fold; this is repeated k times where each fold acts as the test fold. This approach simulates the common learning setting where the training and test sets are disjoint. The resulting k performance estimates are mostly independent, with some dependency introduced due to dependencies between the training sets across the k runs. There is some additional bias introduced from the fact that we do not run the model on the entire training set, but rather get an estimate of the error for the algorithm trained on $n-(n/k)$. For any final models that will be put into production after performing these evaluations, we will likely train on the entire set of n instances.

The bootstrap resample treats the data uses the idea behind bootstrapping: the data constitutes a reasonable model of the data. By sampling from the data, it is like sampling from the distribution that generated the data. To generate training/test splits, the data is sampled with replacement to create the training set, and the remaining unused samples used for test. If k resamples are obtained, we again get k performance measures and can obtain a sample average of performance across different splits and use statistical significance test.

To better understand the properties of these two approaches, see the thorough and accessible explanation in [8, Chapter 5].

10.4 Performance measures for Classification Models

In classification, there are a variety of performance measures to reflect the relative importance of incorrect predictions for either class. For example, it can be more detrimental to predict a patient is not sick if they are actually sick (False Negative), resulting in a decision not to run further diagnostics and so causing serious complications from not treating the illness. When training and evaluating classification algorithms, these preferences need to be encoded. Table 10.3 summarizes some of the terminology for discussing performance of classification models.

Name	Symbol	Definition
Classification error	$error$	$error = \frac{fp+fn}{tp+fp+tn+fn}$
Classification accuracy	$accuracy$	$accuracy = 1 - error$
True positive rate	tpr	$tpr = \frac{tp}{tp+fn}$
False negative rate	fnr	$fnr = \frac{fn}{tp+fn}$
True negative rate	tnr	$tnr = \frac{tn}{tn+fp}$
False positive rate	fpr	$fpr = \frac{fp}{tn+fp}$
Precision	pr	$pr = \frac{tp}{tp+fp}$
Recall	rc	$rc = \frac{tp}{tp+fn}$

Table 10.3: Some classification measures.

Bibliography

[1] P Auer, M Herbster, and Manfred K Warmuth. Exponentially many local minima for single neurons. In *Advances in Neural Information Processing Systems*, 1996.

[2] A Banerjee, S Merugu, I S Dhillon, and J Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 2005.

[3] Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *Journal of Machine Learning Research*, 2002.

[4] Amir Beck and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM J. Imaging Sciences*, 2009.

[5] L on Bottou and Yann Le Cun. On-line learning for very large data sets. *Applied Stochastic Models in Business and Industry*, 2005.

[6] Léon Bottou. Online learning and stochastic approximations. *Online Learning and Neural Networks*, 1998.

[7] Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to Statistical Learning Theory. In *Advanced Lectures on Machine Learning*. Springer Berlin Heidelberg, 2004.

[8] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer New York, 2013.

[9] Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms - A Classification Perspective*. Cambridge University Press, 2011.

[10] Sham M Kakade, Karthik Sridharan, and Ambuj Tewari. On the Complexity of Linear Prediction: Risk Bounds, Margin Bounds, and Regularization. In *Advances in Neural Information Processing Systems*, 2008.

[11] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[12] Lei Le and Martha White. Global optimization of factor models using alternating minimization. *arXiv.org*, 2016.

[13] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Supervised dictionary learning. In *Advances in Neural Information Processing Systems*, 2009.

[14] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 1980.

[15] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research*, 1997.

[16] K B Petersen. The matrix cookbook. *Technical University of Denmark*, 2004.

[17] Dinah Shender and John Lafferty. Computation-Risk Tradeoffs for Covariance-Thresholded Regression. *International Conference on Machine Learning*, 2013.

[18] Ajit P Singh and Geoffrey J Gordon. A unified view of matrix factorization models. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2008.

[19] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2004.

[20] Martha White. *Regularized factor models*. PhD thesis, University of Alberta, 2014.

[21] Matthew D Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv.org*, 2012.

Appendix A

Additional material for probability theory

A.1 Axioms of probability

We can derive other properties from the basic definition of the set of measurable events (the sigma algebra) and probability distributions. The definition of sigma field requires that $\mathcal{E}$ be closed under both finite and countably infinite number of basic set operations (union, intersection, complementation and set difference). The operations union and complementation are in the definition. For intersection, we can use De Morgan's laws: $\cup A_i = (\cap A_i^c)^c$ and $\cap A_i = (\cup A_i^c)^c$. Any intersection of sets in $\mathcal{E}$ must again be in $\mathcal{E}$ because $\mathcal{E}$ is closed under union and complementation. Therefore, a sigma field is also closed under intersection. Similarly for set difference, we can write $A_1 - A_2 = (A_1 \cap A_2)^c \cap A_1$, which then implies $A_1 - A_2 \in \mathcal{E}$ because $A_1 \cap A_2 \in \mathcal{F} \implies (A_1 \cap A_2)^c \in \mathcal{F} \implies (A_1 \cap A_2)^c \cap A_1 \in \mathcal{F}$. Because $\mathcal{E}$ is non-empty, we observe that all the above conditions imply that $\Omega \in \mathcal{E}$ and $\varnothing \in \mathcal{E}$, where $\varnothing$ is the empty set.

For probability distribution $P : \mathcal{E} \to [0, 1]$, we required

1. $P(\Omega) = 1$
2. $A_1, A_2, \ldots \in \mathcal{E}$, $A_i \cap A_j = \varnothing \ \forall i, j \quad \Rightarrow \quad P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$

The tuple $(\Omega, \mathcal{E}, P)$ is called the *probability space*. It seems intuitive that the second condition could be replaced with a union of finite sets (the simpler requirement of additivity rather than σ-additivity). However, for sigma fields, closure under *finite* unions may not result in closure under *infinite* unions.

The beauty of these axioms lies in their compactness and elegance. Many useful expressions can be derived from the axioms of probability. For example, it is obvious that $P(\varnothing) = 0$ or $P(A^c) = 1 - P(A)$. Similarly, closure under infinite unions of disjoint sets (σ-additivity) implies finite closure (additivity), because the remaining sets can be set to the empty set $\varnothing$: $\forall A_1, A_2 \in \mathcal{E}$ with $A_1 \cap A_2 = \varnothing$, set $A_i = \varnothing$ for $i > 2$ to get $P(A_1 \cup A_2) = P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i) = P(A_1) + P(A_2)$. Another formula that is particularly important can be derived by considering a *partition* of the sample space; i.e., a set of k non-overlapping sets $\{B_i\}_{i=1}^k$ such that $\Omega = \cup_{i=1}^k B_i$. That is, if A is any set in Ω and if $\{B_i\}_{i=1}^k$ is a partition of Ω it follows that

$$\begin{aligned} P(A) &= P(A \cap \Omega) \\ &= P\left(A \cap \left(\cup_{i=1}^k B_i\right)\right) \\ &= P\left(\cup_{i=1}^k (A \cap B_i)\right) \\ &= \textstyle\sum_{i=1}^k P\left(A \cap B_i\right), \end{aligned} \tag{A.1}$$

where the last line followed from the axioms of probability. We will refer to this expression as the *sum rule*. Another important expression, shown here without a derivation, is that $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

A.2 A few more useful pmfs

We provide a few more examples of pmfs here, mainly for the purpose that additional concrete examples can be beneficial for understanding. We will not further use these pmfs in these notes.

The *Binomial distribution* is used to describe a sequence of n independent and identically distributed (i.i.d.) Bernoulli trials. At each value k in the sample space the distribution gives the probability that the success happened exactly k times out of n trials, where of course $0 \leq k \leq n$. More formally, given $\Omega = \{0, 1, \ldots, n\}$, for $\forall k \in \Omega$ the binomial pmf is defined as

$$p(k) = \binom{n}{k} \alpha^k (1-\alpha)^{n-k},$$

where $\alpha \in (0, 1)$, as before, is the parameter indicating the probability of success in a single trial. Here, the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

enumerates all ways in which one can pick k elements from a list of n elements (e.g., there are 3 different ways in which one can pick $k = 2$ elements from a group of $n = 3$ elements). We will refer to a binomial distribution with parameters n and α as Binomial(n, α). The experiment leading to a binomial distribution can be generalized to a situation with more than two possible outcomes. This experiment results in a multidimensional probability mass function (one dimension per possible outcome) called the multinomial distribution.

The *geometric distribution* is also used to model a sequence of independent Bernoulli trials with the probability of success α. At each point $k \in \Omega$, it gives the probability that the first success occurs exactly in the k-th trial. Here, $\Omega = \{1, 2, \ldots\}$ and for $\forall k \in \Omega$

$$p(k) = (1-\alpha)^{k-1}\alpha,$$

where $\alpha \in (0, 1)$ is a parameter. The geometric distribution, Geometric(α), is defined over an infinite sample space; i.e., $\Omega = \mathbb{N}$.

For the *hypergeometric distribution*, consider a finite population of N elements of two types (e.g., success and failure), K of which are of one type (e.g., success). The experiment consists of drawing n elements, without replacement, from this population such that the elements remaining in the population are equiprobable in terms of being selected in the next draw. The probability of drawing k successes out of n trials can be described as

$$p(k) = \frac{\binom{K}{k} \cdot \binom{N-K}{n-k}}{\binom{N}{n}},$$

where $0 \leq n \leq N$ and $k \leq n$. The hypergeometric distribution is intimately related to the binomial distribution where the elements are drawn with replacement ($\alpha = {}^{K}/_{N}$). There, the probability of drawing a success does not change in subsequent trials. We will refer to the hypergeometric distribution as Hypergeometric(n, N, K).

A.3 A few more useful pdfs

As mentioned above, we provide a few more pdfs to provide concrete examples, though we will not explicitly use these pdfs in the notes.

The *lognormal distribution* is a modification a normal distribution. Here, for $\Omega = (0, \infty)$ the lognormal density can be expressed as

$$p(\omega) = \frac{1}{\omega\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(\ln\omega - \mu)^2},$$

where $\mu \in \mathbb{R}$ and $\sigma > 0$ are parameters. We will refer to this distribution as Lognormal(μ, σ^2) or $\ln \mathcal{N}(\mu, \sigma^2)$.

The *Gumbel distribution* belongs to the class of extreme value distributions. Its probability density function is defined on $\Omega = \mathbb{R}$ as

$$p(\omega) = \frac{1}{\beta} e^{-\frac{\omega-\alpha}{\beta}} e^{-e^{-\frac{\omega-\alpha}{\beta}}},$$

where $\alpha \in \mathbb{R}$ is the location parameter and $\beta > 0$ is the scale parameter. We will refer to this distribution as Gumbel(α, β).

The *Pareto distribution* is useful for modeling events with rare occurrences of extreme values. Its probability density function is defined on $\Omega = [\omega_{\min}, \infty)$ as

$$p(\omega) = \frac{\alpha \omega_{\min}}{\omega^{\alpha+1}},$$

where $\alpha > 0$ is a parameter and $\omega_{\min} > 0$ is the minimum allowed value for ω. We will refer to the Pareto distribution as Pareto$(\alpha, \omega_{\min})$. It leads to a scale-free property when $\alpha \in (0, 2]$.

A.4 Random Variables

In many situations, we would like to use probabilistic modeling on sets (e.g., a group of people) where elements can be associated with various descriptors. For example, a person may be associated with his/her age, height, citizenship, IQ, or marital status and we may be interested in events related to such descriptors. In other situations, we may be interested in transformations of sample spaces such as those corresponding to digitizing an analog signal from a microphone into a set of integers based on some set of voltage thresholds. The mechanism of a *random variable* facilitates addressing all such situations in a simple, rigorous and unified manner.

A random variable is a variable that, from the observer's point of view, takes values non-deterministically, with generally different preferences for different outcomes. Mathematically, however, it is defined as function that maps one sample space into another, with a few technical caveats we will introduce later. Let us motivate the need for random variables. Consider a probability space $(\Omega, \mathcal{E}, P)$, where Ω is a set of people and let us investigate the probability that a randomly selected person $\omega \in \Omega$ is happy (we may assume we have a diagnostic method to assess any person's status). We start by defining an event A as

$$A = \{\omega \in \Omega : \mathit{Status}(\omega) = \text{happy}\}$$

and simply calculate the probability of this event. This is a perfectly legitimate approach, but it can be much simplified using the random variable mechanism. We first note that, technically, our diagnostic method corresponds to a function $\mathit{Status} : \Omega \to \mathcal{S}$ that maps the sample space Ω to a new binary sample space $\mathcal{S} = \{\text{happy, not happy}\}$. More interestingly, our approach also maps the probability distribution P to a new probability distribution P_{Status} that is defined on some sigma algebra of $\mathcal{S}$; say, $\mathcal{E}_{\mathit{Status}}$ (for the mapping to work as expected, $\mathcal{E}_{\mathit{Status}}$ has to be the power set of $\mathcal{S}$). We can now see that we can calculate $P_{\mathit{Status}}(\{\text{happy}\})$ from the probability of the aforementioned event A; i.e., $P_{\mathit{Status}}(\{\text{happy}\}) = P(A)$. This is a cluttered notation so we may wish to simplify it by using $P(\mathit{Status} = \text{happy})$, where Status is a "random variable".

We will use capital letters $X, Y, \ldots$ to denote random variables (such as Status) and lowercase letters $x, y, \ldots$ to indicate elements (such as "happy") of the new spaces $\mathcal{X}, \mathcal{Y} \ldots$ Generally, we will write probabilities as $P(X = x)$, which is a notational relaxation from $P(\{\omega : X(\omega) = x\})$, or $P(X \leq x)$ for $P(\{\omega : X(\omega) \leq x\})$ when the co-domain $\mathcal{X}$ is continuous. We will also refer to the corresponding probability mass or density functions as $p(x)$ or $p_X(x)$ when when we need to be more explicit about the random variable. This will indeed happen when x takes a particular value; say,

for $x = 1$, we will write $p_X(1)$. Before we proceed to formally define random variables, we shall look at two illustrative examples.

Example 20: [Consecutive tosses of a fair coin.] Consider a process of three coin tosses and two random variables, X and Y, defined on the sample space. We define X as the number of heads in the first toss and Y as the number of heads over all three tosses. Our goal is to find the probability spaces that are created after the transformations.

First, $\Omega = \{\text{HHH}, \text{HHT}, \text{HTH}, \text{HTT}, \text{THH}, \text{THT}, \text{TTH}, \text{TTT}\}$ and

ω	HHH	HHT	HTH	HTT	THH	THT	TTH	TTT
$X(\omega)$	1	1	1	1	0	0	0	0
$Y(\omega)$	3	2	2	1	2	1	1	0

Let us only focus on variable Y. Clearly, $Y : \Omega \to \{0, 1, 2, 3\}$ but we also need to find $\mathcal{E}_Y$ and P_Y. To calculate P_Y, a simple approach is to find its pmf $p(y)$. For example, let us calculate $p_Y(2) = P_Y(\{2\})$ as

$$\begin{aligned} P_Y(\{2\}) &= P(Y = 2) \\ &= P(\{\omega : Y(\omega) = 2\}) \\ &= P(\{\text{HHT}, \text{HTH}, \text{THH}\}) \\ &= \frac{3}{8}, \end{aligned}$$

because of the uniform distribution in the original space $(\Omega, \mathcal{E}, P)$. In a similar way, we can calculate that $P(Y = 0) = P(Y = 3) = 1/8$, and that $P(Y = 1) = 3/8$. In this example, we took that $\mathcal{E} = \mathcal{P}(\Omega)$ and $\mathcal{E}_Y = \mathcal{P}(\mathcal{Y})$. As a final note, we mention that all the randomness is defined in the original probability space $(\Omega, \mathcal{E}, P)$ and that the new probability space $(\mathcal{Y}, \mathcal{E}_Y, P_Y)$ simply inherits it through a deterministic transformation. □

Example 21: [Quantization] Consider $(\Omega, \mathcal{E}, P)$ where $\Omega = [0, 1]$, $\mathcal{E} = \mathcal{B}(\Omega)$, and P is induced by a uniform pdf. Define $X : \Omega \to \{0, 1\}$ as

$$X(\omega) = \begin{cases} 0 & \omega \leq 0.5 \\ 1 & \omega > 0.5 \end{cases}$$

and find the transformed probability space.

Technically, we have changed the sample space to $\mathcal{X} = \{0, 1\}$. For an event space $\mathcal{E}_X = \mathcal{P}(\mathcal{X}) = \{\varnothing, \{0\}, \{1\}, \{0, 1\}\}$ we would like to understand the new probability distribution P_X. We have

$$\begin{aligned} p_X(0) &= P_X(\{0\}) \\ &= P(X = 0) \\ &= P(\{\omega : \omega \in [0, 0.5]\}) \\ &= \frac{1}{2} \end{aligned}$$

and

$$\begin{aligned} p_X(1) &= P_X(\{1\}) \\ &= P(X = 1) \\ &= P(\{\omega : \omega \in (0.5, 1]\}) \\ &= \frac{1}{2} \end{aligned}$$

From here we can easily see that $P_X(\{0, 1\}) = 1$ and $P_X(\varnothing) = 0$, and so P_X is indeed a probability distribution. Again, P_X is naturally defined using P. Thus, we have transformed the probability space $(\Omega, \mathcal{E}, P)$ into $(\mathcal{X}, \mathcal{E}_X, P_X)$. □

A.4.1 Formal definition of random variable

We now formally define a random variable. Given a probability space $(\Omega, \mathcal{E}, P)$, a random variable X is a function $X : \Omega \to \mathcal{X}$ such that for every $A \in \mathcal{B}(\mathcal{X})$ it holds that $\{\omega : X(\omega) \in A\} \in \mathcal{E}$. It follows that

$$P_X(A) = P(\{\omega : X(\omega) \in A\}).$$

It is important to mention that, by default, we defined the event space of a random variable to be the Borel field of $\mathcal{X}$. This is convenient because a Borel field of a countable set Ω is its power set. Thus, we are working with the largest possible event spaces for both discrete and continuous random variables.

Consider now a *discrete random variable* X defined on $(\Omega, \mathcal{E}, P)$. As we can see from the previous examples, the probability distribution for X can be found as

$$\begin{aligned} p(x) &= P_X(\{x\}) \\ &= P(\{\omega : X(\omega) = x\}) \end{aligned}$$

for $\forall x \in \mathcal{X}$. The probability of an event A can be found as

$$\begin{aligned} P_X(A) &= P(\{\omega : X(\omega) \in A\}) \\ &= \sum_{x \in A} p(x) \end{aligned}$$

for $\forall A \subseteq \mathcal{X}$.

The case of *continuous random variables* is more complicated, but reduces to an approach that is similar to that of discrete random variables. Here we first define a *cumulative distribution function* (cdf) as

$$\begin{aligned} F_X(t) &= P_X(\{x : x \leq t\}) \\ &= P(\{\omega : X(\omega) \leq t\}) \\ &= P(X \leq t), \end{aligned}$$

where $P(X \leq t)$, as before, presents a minor abuse of notation. If the cumulative distribution function is differentiable, the probability density function of a continuous random variable is defined as

$$p(x) = \left. \frac{dF_X(t)}{dt} \right|_{t=x}.$$

Alternatively, if $p(x)$ exists, then

$$F_X(t) = \int_{-\infty}^{t} p(x)\, dx,$$

for each $t \in \mathbb{R}$. Our focus will be exclusively on random variables that have their probability density functions; however, for a more general view, we should always keep in mind "if one exists" when referring to pdfs.

The probability that a random variable will take a value from interval $(a, b]$ can now be calculated as

$$\begin{aligned} P_X((a, b]) &= P(a < X \leq b) \\ &= \int_a^b p(x)\, dx \\ &= F_X(b) - F_X(a), \end{aligned}$$

which follows from the properties of integration.

Suppose now that random variable X transforms probability space $(\Omega, \mathcal{E}, P)$ into $(\mathcal{X}, \mathcal{B}(\mathcal{X}), P_X)$. To describe the resulting probability space, we commonly use probability mass and density functions inducing P_X. For example, if P_X is induced by a Gaussian distribution with parameters μ and σ^2, we use

$$X : \mathcal{N}(\mu, \sigma^2) \qquad \text{or} \qquad X \sim \mathcal{N}(\mu, \sigma^2).$$

Both notations indicate that the probability density function for the random variable X is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}.$$

The Gaussian density implicitly defined that $\mathcal{X} = \mathbb{R}$. This point however is superficial because we can always extend the domain of a density function to $\mathbb{R}$ and set $p(x) = 0$ wherever the original function was not defined.

A group of d random variables $\{X_i\}_{i=1}^d$ defined on the same probability space $(\Omega, \mathcal{E}, P)$ is called a *random vector* or a multivariate (multidimensional) random variable. We have already seen an example of a random vector provided by random variables (X, Y) in Example 20. A generalization of a random vector to infinite sets is referred to as a *random process* or *stochastic process*; i.e., $\{X_i : i \in \mathcal{T}\}$, where $\mathcal{T}$ is an index set usually interpreted as a set of time indices. In the case of discrete time indices (e.g., $\mathcal{T} = \mathbb{N}$) the random process is called a discrete-time random process; otherwise (e.g., $\mathcal{T} = \mathbb{R}$) it is called a continuous-time random process. There are many models in machine learning that deal with temporally-connected random variables (e.g., autoregressive models for time series, Markov chains, hidden Markov models, dynamic Bayesian networks). The language of random variables, through stochastic processes, nicely enables formalization of these models. Most of these notes, however, will deal with simpler settings only requiring (i.i.d.) multivariate random variables.

Example 22: [Three tosses of a fair coin for joint probabilities.] Consider the two random variables from Example 20 and calculate their probability spaces, joint and marginal distributions. Recall X is the number of heads in the first toss and Y is the number of heads over all three tosses.

A joint probability mass function $p(x, y) = P(X = x, Y = y)$ is shown below

		Y			
		0	1	2	3
X	0	$1/8$	$1/4$	$1/8$	0
	1	0	$1/8$	$1/4$	$1/8$

but let us step back for a moment and show how we can calculate it. Let us consider two sets $A = \{\,\text{HHH, HHT, HTH, HTT}\,\}$ and $B = \{\,\text{HHT, HTH, THH}\,\}$, corresponding to the events that the first toss was heads and that there were exactly two heads over the three tosses, respectively. Now, let us look at the probability of the intersection of A and B

$$\begin{aligned} P(A \cap B) &= P(\{\text{HHT}, \text{HTH}\}) \\ &= \frac{1}{4} \end{aligned}$$

We can represent the probability of the logical statement $X = 1 \wedge Y = 2$ as

$$\begin{aligned} p_{XY}(1, 2) &= P(X = 1, Y = 2) \\ &= P(A \cap B) \\ &= P(\{\text{HHT}, \text{HTH}\}) \\ &= \frac{1}{4}. \end{aligned}$$

The marginal probability distribution can be found in a straightforward way as

$$p(x) = \sum_{y \in \mathcal{Y}} p(x, y),$$

where $\mathcal{Y} = \{0, 1, 2, 3\}$. Thus,

$$\begin{aligned} p_X(0) &= \sum_{y \in \mathcal{Y}} p_{XY}(0, y) \\ &= \frac{1}{2}. \end{aligned}$$

We note for the end that in the discrete case we have $|\mathcal{X}| \cdot |\mathcal{Y}| - 1$ free parameters (because the sum must equal 1) to fully describe the joint distribution $p(x, y)$. Asymptotically, this corresponds to an exponential growth of the number of entries in the table with the number of random variables (d). For example, if $|\mathcal{X}_i| = 2$ for $\forall X_i$, there are $2^d - 1$ free elements in the joint probability distribution. Estimating such distributions from data is intractable and is one form of the *curse of dimensionality*. □

A.4.2 Example of conditional independence

We show that independence and conditional independence do not imply each in other, in two simple examples from Figure A.1. This example is mostly algebraic, and is a useful exercise to indicate this property, but does not provide much intuition about why this occurs. The d-separation rules given in Section A.6 further explain why you can have these different dependencies. We have already provided an example of X and Y that are conditionally independent given Z, but not independent, in Example 7. We provide one more example here for when two variables X and Y are independent, but not conditionally independent given Z. The key is that the information in Z couples X and Y, whereas in Example 7, knowing Z (the bias of the coin) decoupled X and Y (two flips of the biased coin).

Example 23: Consider a setting where you try to predict the price of a house. You have many samples of previous house prices, but without any associated features. Let X and Y be the price of two different houses. Without any additional information, these two variables are independent—in fact, we can consider them as i.i.d. samples from some underlying distribution over house prices. However, if we are now given an addition piece of information that both houses share, then they become dependent. Let Z correspond to the variable that the two houses are in the same neighborhood (i.e., a 0 or 1 variable). If $Z = 1$, then knowing the price of X definitely influences what the distribution over prices are for Y. The addition of this feature makes these two random variables conditionally dependent. □

A.4.3 Additional information for expectations and moments

In this section, we provide a few additional examples of expectations of functions of a random variable that are often considered—enough so to be given names. Recall that for a function $f : \mathcal{X} \to \mathbb{R}$, we have the expected value $\mathbb{E}[f(X)] = \sum f(x)p(x)$. Using $f(x) = x^k$ results in the k-th moment, $f(x) = \log {}^1\!/_{p(x)}$ gives the well-known entropy function $H(X)$, or differential entropy for continuous random variables, and $f(x) = (x - \mathbb{E}[X])^2$ provides the variance of a random variable X, denoted by $V[X]$. Interestingly, the probability of some event $A \subseteq \mathcal{X}$ can also be expressed in the form of expectation; i.e.,

$$P(A) = \mathbb{E}[1(X \in A)],$$

A. X and Y are independent, but not conditionally independent given Z

$P(X=1)$
a

X	$P(Y=1\|X)$
0	b
1	b

X	Y	$P(Z=1\|X,Y)$
0	0	c
0	1	$1-c$
1	0	$1-c$
1	1	c

$$P(Y=y|X=x) = P(Y=y)$$

for example,

$$P(Y=1|X=x) = b$$
$$P(Y=1) = b$$

$$P(Y=y|X=x, Z=z) \neq P(Y=y|Z=z)$$

for example,

$$P(Y=1|X=1, Z=1) = bc/(1-c-b(1-2c))$$
$$P(Y=1|Z=1) = b(1-c-a(1-2c))/d$$
where $d = P(Z=1)$

B. X and Z are conditionally independent given Y, but not independent

$P(X=1)$
a

X	$P(Y=1\|X)$
0	b
1	c

X	Y	$P(Z=1\|X,Y)$
0	0	d
0	1	e
1	0	d
1	1	e

$$P(Z=z|X=x) \neq P(Z=z)$$

for example,

$$P(Z=1|X=1) = d + ce - cd$$
$$P(Z=1) = d + (e-d)(a(c-b)+b)$$

$$P(Z=z|X=x, Y=y) = P(Z=z|Y=y)$$

for example,

$$P(Z=1|X=x, Y=1) = e$$
$$P(Z=1|Y=1) = e$$

Figure A.1: Independence vs. conditional independence using probability distributions involving three binary random variables. Probability distributions are presented using factorization $p(x,y,z) = p(x)p(y|x)p(z|x,y)$, where all constants $a,b,c,d,e \in [0,1]$. (A) Variables X and Y are independent, but not conditionally independent given Z. When $c=0$, $Z = X \oplus Y$, where $\oplus$ is an "exclusive or" operator. (B) Variables X and Z are conditionally independent given Y, but are not independent.

$f(x)$	Symbol	Name
x	$\mathbb{E}[X]$	Mean
$(x - \mathbb{E}[X])^2$	$V[X]$	Variance
x^k	$\mathbb{E}[X^k]$	k-th moment; $k \in \mathbb{N}$
$(x - \mathbb{E}[X])^k$	$\mathbb{E}[(X - \mathbb{E}[X])^k]$	k-th central moment; $k \in \mathbb{N}$
e^{tx}	$M_X(t)$	Moment generating function
e^{itx}	$\varphi_X(t)$	Characteristic function
$\log \frac{1}{p(x)}$	$H(X)$	(Differential) entropy
$\log \frac{p(x)}{q(x)}$	$D(p\|\|q)$	Kullback-Leibler divergence
$\left(\frac{\partial}{\partial\theta} \log p(x\|\theta)\right)^2$	$\mathcal{I}(\theta)$	Fisher information

Table A.1: Some important expectation functions $\mathbb{E}[f(X)]$ for a random variable X described by its distribution $p(x)$. Function $q(x)$ in the definition of the Kullback-Leibler divergence is non-negative and must sum (integrate) to 1; i.e., it is a probability distribution itself. The Fisher information is defined for a family of probability distributions specified by a parameter θ. Note that the moment generating function may not exist for some distributions and all values of t; however, the characteristic function always exists, even when the density function does not.

where

$$1(t) = \begin{cases} 1 & t \text{ is true} \\ 0 & t \text{ is false} \end{cases} \tag{A.2}$$

is an indicator function. With this, it is possible to express the cumulative distribution function as $F_X(t) = \mathbb{E}[1(X \in (-\infty, t])]$.

Function $f(x)$ inside the expectation can also be complex-valued. For example, $\varphi_X(t) = \mathbb{E}[e^{itX}]$, where i is the imaginary unit, defines the characteristic function of X. The characteristic function is closely related to the inverse Fourier transform of $p(x)$ and is useful in many forms of statistical inference. Several expectation functions are summarized in Table A.1.

Example 24: [Three tosses of a fair coin (yet again).] Consider two random variables from Examples 3 and 5, and calculate the expectation and variance for both X and Y. Then calculate $\mathbb{E}[Y|X = 0]$.

We start by calculating $\mathbb{E}[X] = 0 \cdot p_X(0) + 1 \cdot p_X(1) = \frac{1}{2}$. Similarly,

$$\begin{aligned} \mathbb{E}[Y] &= \sum_{y=0}^{3} y \cdot p_Y(y) \\ &= p_Y(1) + 2p_Y(2) + 3p_Y(3) \\ &= \frac{3}{2} \end{aligned}$$

$f(x,y)$	Symbol	Name
$(x - \mathbb{E}[X])(y - \mathbb{E}[Y])$	$\mathrm{Cov}[X,Y]$	Covariance
$\frac{(x-\mathbb{E}[X])(y-\mathbb{E}[Y])}{\sqrt{V[X]V[Y]}}$	$\mathrm{Corr}[X,Y]$	Correlation
$\log \frac{p(x,y)}{p(x)p(y)}$	$I(X;Y)$	Mutual information
$\log \frac{1}{p(x,y)}$	$H(X,Y)$	Joint entropy
$\log \frac{1}{p(x\|y)}$	$H(X\|Y)$	Conditional entropy

Table A.2: Some important expectation functions $\mathbb{E}[f(X,Y)]$ for two random variables, X and Y, described by their joint distribution $p(x,y)$. Mutual information is sometimes referred to as average mutual information*.*

The conditional expectation can be found as

$$\begin{aligned}\mathbb{E}[Y|X=0] &= \sum_{y=0}^{3} y \cdot p_{Y|X}(y|0) \\ &= p_{Y|X}(1|0) + 2p_{Y|X}(2|0) + 3p_{Y|X}(3|0) \\ &= 1\end{aligned}$$

where $p(y|x) = p(x,y)/p(x)$.

□

A.5 Mixtures of distributions

In previous sections we saw that random variables are often described using particular families of probability distributions. This approach can be generalized by considering mixtures of distributions; i.e., linear combinations of other probability distributions. As before, we shall only consider random variables that have their probability mass or density functions.

Given a set of m probability distributions, $\{p_i(x)\}_{i=1}^m$, a finite mixture distribution function, or *mixture model*, $p(x)$ is defined as

$$p(x) = \sum_{i=1}^{m} w_i p_i(x), \tag{A.3}$$

where $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)$ is a set of non-negative real numbers such that $\sum_{i=1}^m w_i = 1$. We refer to $\boldsymbol{w}$ as mixing coefficients or, sometimes, as mixing probabilities. A linear combination with such coefficients is called a convex combination. It is straightforward to verify that a function defined in this manner is indeed a probability distribution.

Here we will briefly look into the basic expectation functions of the mixture distribution. Suppose $\{X_i\}_{i=1}^m$ is a set of m random variables described by their respective probability distributions $\{p_{X_i}(x)\}_{i=1}^m$. Suppose also that a random variable X is described by a mixture distribution with coefficients $\boldsymbol{w}$ and probability distributions $\{p_{X_i}(x)\}_{i=1}^m$. Then, assuming continuous random variables

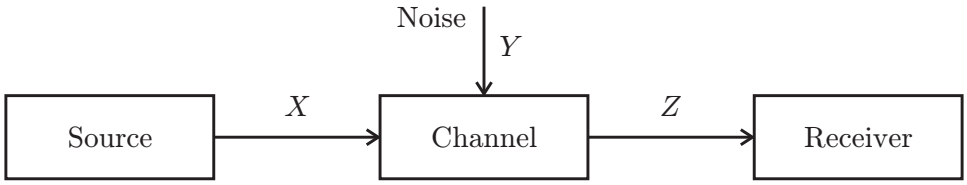


Figure A.2: A digital signal communication system with additive noise.

defined on $\mathbb{R}$, the expectation function is given as

$$\begin{aligned}\mathbb{E}\left[f(X)\right] &= \int_{-\infty}^{+\infty} f(x)p_X(x)dx \\ &= \int_{-\infty}^{+\infty} f(x)\sum_{i=1}^{m} w_i p_{X_i}(x)dx \\ &= \sum_{i=1}^{m} w_i \int_{-\infty}^{+\infty} f(x)p_{X_i}(x)dx \\ &= \sum_{i=1}^{m} w_i \mathbb{E}[f(X_i)].\end{aligned}$$

We can now apply this formula to obtain the mean, when $f(x) = x$ and the variance, when $f(x) = (x - E[X])^2$, of the random variable X as

$$\mathbb{E}[X] = \sum_{i=1}^{m} w_i \mathbb{E}[X_i],$$

and

$$V[X] = \sum_{i=1}^{m} w_i V[X_i] + \sum_{i=1}^{m} w_i \left(\mathbb{E}[X_i] - \mathbb{E}[X]\right)^2,$$

respectively.

Example 25: Signal communications. Consider transmission of a single binary digital signal (bit) over a noisy communication channel shown in Figure A.2. The magnitude of the signal X emitted by the source is equally likely to be 0 or 1 Volt. The signal is sent over a transmission line (e.g., radio communication, optical fiber, magnetic tape) in which a zero-mean normally distributed noise component Y is added to X. Derive the probability distribution of the signal $Z = X + Y$ that enters the receiver.

We will consider a slightly more general situation where X : Bernoulli(α) and Y : Gaussian(μ, σ^2). To find $p(z)$ we will use characteristic functions of random variables X, Y and Z, written as $\varphi_X(t) = \mathbb{E}[e^{itX}]$, $\varphi_Y(t) = \mathbb{E}[e^{itY}]$ and $\varphi_Z(t) = \mathbb{E}[e^{itZ}]$. Without derivation we write

$$\varphi_X(t) = 1 - \alpha + \alpha e^{it}$$
$$\varphi_Y(t) = e^{it\mu - \frac{\sigma^2 t^2}{2}}$$

and subsequently

$$\begin{aligned}\varphi_Z(t) &= \varphi_{X+Y}(t) \\ &= \varphi_X(t) \cdot \varphi_Y(t) \\ &= \left(1-\alpha+\alpha e^{it}\right) \cdot e^{it\mu-\frac{\sigma^2 t^2}{2}} \\ &= \alpha e^{it(\mu+1)-\frac{\sigma^2 t^2}{2}} + (1-\alpha)e^{it\mu-\frac{\sigma^2 t^2}{2}}.\end{aligned}$$

By performing integration on $\varphi_Z(t)$ we can easily verify that

$$p(z) = \alpha \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(z-\mu-1)^2} + (1-\alpha) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(z-\mu)^2},$$

which is a mixture of two normal distributions $\mathcal{N}(\mu+1, \sigma^2)$ and $\mathcal{N}(\mu, \sigma^2)$ with coefficients $w_1 = \alpha$ and $w_2 = 1-\alpha$, respectively. Observe that a convex combination of random variables $Z = w_1 X + w_2 Y$ does not imply $p_Z(x) = w_1 p_X(x) + w_2 p_Y(x)$.

□

A.6 Graphical representation of probability distributions

We saw earlier that a joint probability distribution can be *factorized* using the chain rule from Equation (1.2). Such factorizations can be visualized using a directed graph representation, where nodes represent random variables and edges depict dependence. For example,

$$p(x, y, z) = p(x)p(y|x)p(z|x, y)$$

is shown in Figure A.3A. Graphical representations of probability distributions using directed acyclic graphs, together with conditional probability distributions, are called *Bayesian networks* or *belief networks*. They facilitate interpretation as well as effective statistical inference.

Visualizing relationships between variables becomes particularly convenient when we want to understand and analyze conditional independence properties of variables. Figure A.3B shows the the same factorization of $p(x, y, z)$ where variable Z is independent of X given Y. To carefully determine conditional independence and dependence properties, however, one usually uses the *d-separation* rules for belief networks. Though often relationships are intuitive, sometimes dependence properties can get more complicated due to multiple relationships between nodes. For example, in Figure A.4A, two nodes do not have an edge, but are conditionally dependent through another node. On the other hand, in Figure A.4B, the absence of an edge does imply conditional independence. We will not further examine d-separation rules at this time; they can easily be found in any standard textbook on graphical models.

Belief networks have a simple, formal definition. Given a set of d random variables $\boldsymbol{X} = (X_1, \ldots, X_d)$, belief networks factorize the joint probability distribution of $\boldsymbol{X}$ as

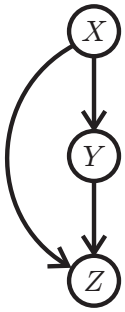
$$p(\boldsymbol{x}) = \prod_{i=1}^{d} p\left(x_i | \boldsymbol{x}_{\text{Parents}(X_i)}\right),$$

where $\text{Parents}(X)$ denotes the immediate ancestors of node X in the graph. In Figure A.3B, node Y is a parent of Z, but node X is not a parent of Z.

It is important to mention that there are multiple (how many?) ways of factorizing a distribution. For example, by reversing the order of variables $p(x, y, z)$ can be also factorized as

$$p(x, y, z) = p(z)p(y|z)p(x|y, z),$$

A. Discrete probability distribution without conditional independences



$P(X=1)$
0.3

X	$P(Y=1\|X)$
0	0.5
1	0.9

X	Y	$P(Z=1\|X,Y)$
0	0	0.3
0	1	0.1
1	0	0.7
1	1	0.4

$$P(X=x, Y=y, Z=z) = P(X=x)P(Y=y|X=x)P(Z=z|X=x, Y=y)$$

B. Discrete probability distribution; Z is conditionally independent of X given Y

$P(X=1)$
0.3

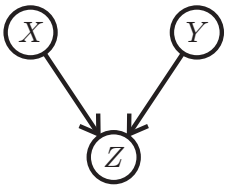
X	$P(Y=1\|X)$
0	0.5
1	0.9

Y	$P(Z=1\|Y)$
0	0.2
1	0.7

$$P(X=x, Y=y, Z=z) = P(X=x)P(Y=y|X=x)P(Z=z|Y=y)$$

Figure A.3: Bayesian network: graphical representation of two joint probability distributions for three discrete (binary) random variables (X, Y, Z) using directed acyclic graphs. The probability mass function $p(x, y, z)$ is defined over $\{0, 1\}^3$. (A) Full factorization; (B) Factorization that shows and ensures conditional independence between Z and X, given Y. Each node is associated with a conditional probability distribution. In discrete cases, these conditional distributions are referred to as conditional probability tables.

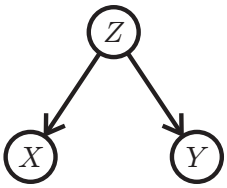
A. X is independent of Y, but not given Z



$$P(X = x \,|\, Y = y) = P(X = x)$$

$$P(X = x \,|\, Y = y, Z = z) \neq P(X = x \,|\, Z = z)$$

B. X and Y are dependent, but conditionally independent given Z



$$P(X = x \,|\, Y = y, Z = z) = P(X = x \,|\, Z = z)$$

$$P(Y = y \,|\, X = x, Z = z) = P(Y = y \,|\, Z = z)$$

Figure A.4: Two examples of Bayesian networks. (A) A model where the lack of an edge between nodes does not indicate independence. Given information about Z, X and Y are actually dependent; i.e., they are conditionally dependent through Z. (B) A model where the lack of an edge between nodes does indicate independence. Given information about Z, X and Y are conditionally independent. We will see this representation later under Naive Bayes models.

which has a different graphical representation and its own conditional probability distributions, yet the same joint probability distribution as the earlier factorization. Selecting a proper factorization and estimating the conditional probability distributions from data will be discussed in detail later.

Undirected graphs can also be used to factorize probability distributions. The main idea here is to decompose graphs into maximal cliques $\mathcal{C}$ (the smallest set of cliques that covers the graph) and express the distribution in the following form

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C),$$

where each $\psi_C(\boldsymbol{x}_C) \geq 0$ is called the clique potential function and

$$Z = \int_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C) d\boldsymbol{x},$$

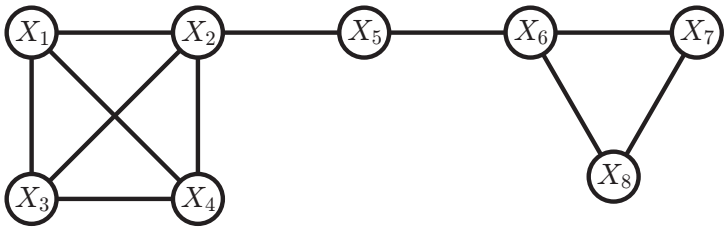
is called the partition function, used strictly for normalization purposes. In contrast to conditional probability distributions in directed acyclic graphs, the clique potentials usually do not have conditional probability interpretations and, thus, normalization is necessary. One example of a maximum clique decomposition is shown in Figure A.5.

The potential functions are typically taken to be strictly positive, $\psi_C(\boldsymbol{x}_C) > 0$, and expressed as

$$\psi_C(\boldsymbol{x}_C) = \exp\left(-E(\boldsymbol{x}_C)\right),$$

where $E(\boldsymbol{x}_C)$ is a user-specified energy function on the clique of random variables $\boldsymbol{X}_C$. This leads to the probability distribution of the following form

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp\left(\sum_{C \in \mathcal{C}} \log \psi_C(\boldsymbol{x}_C)\right).$$



$$\boldsymbol{X}_{C_1} = \{X_1, X_2, X_3, X_4\} \qquad \boldsymbol{X}_{C_3} = \{X_5, X_6\}$$
$$\boldsymbol{X}_{C_2} = \{X_2, X_5\} \qquad \boldsymbol{X}_{C_4} = \{X_6, X_7, X_8\}$$

Figure A.5: Markov network: graphical representation of a probability distribution using maximum clique decomposition. Shown is a set of eight random variables with their interdependency structure and maximum clique decomposition (a clique is fully connected subgraph of a given graph). A decomposition into maximum cliques covers all vertices and edges in a graph with the minimum number of cliques. Here, the set of variables is decomposed into four maximal cliques $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$.

As formulated, this probability distribution is called the Boltzmann distribution or the Gibbs distribution.

The energy function $E(\boldsymbol{x})$ must be lower for values of $\boldsymbol{x}$ that are more likely. It also may involve parameters that are then estimated from the available training data. Of course, in a prediction problem, an undirected graph must be created to also involve the target variables, which were here considered to be a subset of $\boldsymbol{X}$.

Consider now any probability distribution over all possible configurations of the random vector $\boldsymbol{X}$ with its underlying graphical representation. If the following property

$$p\left(x_i|\boldsymbol{x}_{-X_i}\right) = p\left(x_i|\boldsymbol{x}_{N(X_i)}\right) \tag{A.4}$$

is satisfied, the probability distribution is referred to as *Markov network* or a *Markov random field*. In the equation above

$$\boldsymbol{X}_{-X_i} = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_d)$$

and $N(X)$ is a set of random variables neighboring X in the graph; i.e., there exists an edge between X and every node in $N(X)$. The set of random variables in $N(X)$ is also called the Markov blanket of X.

It can be shown that every Gibbs distribution satisfies the property from Equation (A.4) and, conversely, that for every probability distribution for which Equation (A.4) holds can be represented as a Gibbs distribution with some choice of parameters. This equivalence of Gibbs distributions and Markov networks was established by the Hammersley-Clifford theorem.

Appendix B

Optimization background

B.1 Basic rules for gradients

For derivatives, there are useful rules that you are familiar with, such as $\frac{d}{dw}aw = a$, $\frac{d}{dw}w^2 = 2w$ and $\frac{d}{dw}e^w = e^w$. We can similarly write down such rules for the multivariate setting, to simplify computation of gradients without having to go resort to computing each partial derivative. Each of the following rules can be verified by computing partial derivatives, with the rules you are used to for the univariate case. We summarize the key rules for this document here; for a more complete reference, see the matrix cookbook [16].

Some of these rules are summarized in Table B.1. This list is not comprehensive, but does enable some additional rules to be derived. For example, to obtain the derivative for the function $f(\mathbf{x}) = \mathbf{x}^\top\mathbf{A}$, one can first obtain the derivative for $f(\mathbf{x})^\top = \mathbf{A}^\top\mathbf{x}$ and then take its transpose because

$$\big(\nabla f(\mathbf{x})\big)^\top = \nabla\big(f(\mathbf{x})^\top\big).$$

Therefore, because $\nabla f(\mathbf{x})^\top = \mathbf{A}^\top$, we get that $\nabla f(\mathbf{x}) = \mathbf{A}$.

$f(\mathbf{x})$	$\frac{\partial f}{\partial \mathbf{x}}$
$\mathbf{x}^\top\mathbf{x}$	$2\mathbf{x}$
$\mathbf{A}\mathbf{x}$	$\mathbf{A}$
$\mathbf{x}^\top\mathbf{A}\mathbf{x}$	$\mathbf{A}\mathbf{x} + \mathbf{A}^\top\mathbf{x}$

Table B.1: Useful derivative formulas of vectors with respect to vectors. The derivative of vector-valued function $f : \mathbb{R}^{d\times 1} \to \mathbb{R}^{m\times 1}$ with respect to vector $\mathbf{x} \in \mathbb{R}^{d\times 1}$ is an $m \times d$ matrix $\mathbf{M}$ with components $M_{ij} = \partial y_j/\partial x_i$, $i \in \{1, 2, \ldots, d\}$ and $j \in \{1, 2, \ldots, m\}$. A derivative of scalar with respect to a vector, where $m = 1$, is a special case of this situation that results in an $d \times 1$ column vector. Note that in the table, m is not the same for each row. For example, $f(\mathbf{x}) = \mathbf{x}^\top\mathbf{x}$ is a scalar, whereas for a general matrix $\mathbf{A} \in \mathbb{R}^{m\times d}$, $f(\mathbf{x}) = \mathbf{A}\mathbf{x}$ is a m-dimensional vector.

Appendix C

Linear algebra background

C.1 An Algebraic Perspective

Another powerful tool for analyzing and understanding linear regression comes from linear and applied linear algebra. In this section we take a detour to address fundamentals of linear algebra and then apply these concepts to deepen our understanding of regression. In linear algebra, we are frequently interested in solving the following set of equations, given below in a matrix form

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{C.1}$$

Here, $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{b}$ is an $m \times 1$ vector, and $\mathbf{x}$ is an $n \times 1$ vector that is to be found. All elements of $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$ are considered to be real numbers. We shall start with a simple scenario and assume $\mathbf{A}$ is a square, 2×2 matrix. This set of equations can be expressed as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned}$$

For example, we may be interested in solving

$$\begin{aligned} x_1 + 2x_2 &= 3 \\ x_1 + 3x_2 &= 5 \end{aligned}$$

This is a convenient formulation when we want to solve the system, e.g. by Gaussian elimination. However, it is not a suitable formulation to understand the question of the existence of solutions. In order for us to do this, we briefly review the basic concepts in linear algebra.

C.1.1 The four fundamental subspaces

The objective of this section it to briefly review the *four fundamental subspaces* in linear algebra (column space, row space, nullspace, left nullspace) and their mutual relationship. We shall start with our example from above and write the system of linear equations as

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} x_1 + \begin{bmatrix} 2 \\ 3 \end{bmatrix} x_2 = \begin{bmatrix} 3 \\ 5 \end{bmatrix}.$$

We can see now that by solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ we are looking for the right amounts of vectors $(1, 1)$ and $(2, 3)$ so that their linear combination produces $(3, 5)$; these amounts are $x_1 = -1$ and $x_2 = 2$. Let us define $\mathbf{a}_1 = (1, 1)$ and $\mathbf{a}_2 = (2, 3)$ to be the column vectors of $\mathbf{A}$; i.e. $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2]$. Thus, $\mathbf{A}\mathbf{x} = \mathbf{b}$ will be solvable whenever $\mathbf{b}$ can be expressed as a linear combination of the column vectors $\mathbf{a}_1$ and $\mathbf{a}_2$.

All linear combinations of the columns of matrix $\mathbf{A}$ constitute the *column space* of $\mathbf{A}$, $C(\mathbf{A})$, with vectors $\mathbf{a}_1 \ldots \mathbf{a}_n$ being a basis of this space. Both $\mathbf{b}$ and $C(\mathbf{A})$ lie in the m-dimensional space $\mathbb{R}^m$. Therefore, what $\mathbf{A}\mathbf{x} = \mathbf{b}$ is saying is that $\mathbf{b}$ must lie in the column space of $\mathbf{A}$ for the equation to have

solutions. In the example above, if columns of $\mathbf{A}$ are linearly independent[1], the solution is unique, i.e. there exists only one linear combination of the column vectors that will give $\mathbf{b}$. Otherwise, because $\mathbf{A}$ is a square matrix, the system has no solutions. An example of such a situation is

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix},$$

where $\mathbf{a}_1 = (1, 1)$ and $\mathbf{a}_2 = (2, 2)$. Here, $\mathbf{a}_1$ and $\mathbf{a}_2$ are (linearly) dependent because $2\mathbf{a}_1 - \mathbf{a}_2 = 0$. There is a deep connection between the spaces generated by a set of vectors and the properties of the matrix A. For now, using the example above, it suffices to say that if $\mathbf{a}_1$ and $\mathbf{a}_2$ are independent the matrix $\mathbf{A}$ is non-singular (singularity can be discussed only for square matrices), that is of full rank.

In an equivalent manner to the column space, all linear combinations of the rows of $\mathbf{A}$ constitute the *row space*, denoted by $C(\mathbf{A}^\top)$, where both $\mathbf{x}$ and $C(\mathbf{A}^\top)$ are in $\mathbb{R}^n$. All solutions to $\mathbf{Ax} = \mathbf{0}$ constitute the *nullspace* of the matrix, $N(\mathbf{A})$, while all solutions of $\mathbf{A}^\top\mathbf{y} = \mathbf{0}$ constitute the so-called *left nullspace* of $\mathbf{A}$, $N(\mathbf{A}^\top)$. Clearly, $C(\mathbf{A})$ and $N(\mathbf{A}^\top)$ are embedded in $\mathbb{R}^m$, whereas $C(\mathbf{A}^\top)$ and $N(\mathbf{A})$ are in $\mathbb{R}^n$. However, the pairs of subspaces are orthogonal (vectors $\mathbf{u}$ and $\mathbf{v}$ are orthogonal if $\mathbf{u}^\top\mathbf{v} = 0$); that is, any vector in $C(\mathbf{A})$ is orthogonal to all vectors from $N(\mathbf{A}^\top)$ and any vector in $C(\mathbf{A}^\top)$ is orthogonal to all vectors from $N(\mathbf{A})$. This is easy to see: if $\mathbf{x} \in N(\mathbf{A})$, then by definition $\mathbf{Ax} = \mathbf{0}$, and thus each row of $\mathbf{A}$ is orthogonal to $\mathbf{x}$. If each row is orthogonal to $\mathbf{x}$, then so are all linear combinations of rows.

Orthogonality is a key property of the four subspaces, as it provides useful decomposition of vectors $\mathbf{x}$ and $\mathbf{b}$ from Eq. (C.1) with respect to $\mathbf{A}$ (we will exploit this in the next Section). For example, any $\mathbf{x} \in \mathbb{R}^n$ can be decomposed as

$$\mathbf{x} = \mathbf{x}_r + \mathbf{x}_n,$$

where $\mathbf{x}_r \in C(\mathbf{A}^\top)$ and $\mathbf{x}_n \in N(\mathbf{A})$, such that $\|\mathbf{x}\|_2^2 = \|\mathbf{x}_r\|_2^2 + \|\mathbf{x}_n\|_2^2$. Similarly, every $\mathbf{b} \in \mathbb{R}^m$ can be decomposed as

$$\mathbf{b} = \mathbf{b}_c + \mathbf{b}_l,$$

where $\mathbf{b}_c \in C(\mathbf{A})$, $\mathbf{b}_l \in N(\mathbf{A}^\top)$, and $\|\mathbf{b}\|_2^2 = \|\mathbf{b}_c\|_2^2 + \|\mathbf{b}_l\|_2^2$.

We mentioned above that the properties of fundamental spaces are tightly connected with the properties of matrix $\mathbf{A}$. To conclude this section, let us briefly discuss the *rank* of a matrix and its relationship with the dimensions of the fundamental subspaces. The *basis* of the space is the smallest set of vectors that span the space (this set of vectors is not unique). The size of the basis is also called the dimension of the space. In the example at the beginning of this subsection, we had a two dimensional column space with basis vectors $\mathbf{a}_1 = (1, 1)$ and $\mathbf{a}_2 = (2, 3)$. On the other hand, for $\mathbf{a}_1 = (1, 1)$ and $\mathbf{a}_2 = (2, 2)$ we had a one dimensional column space, i.e. a line, fully determined by any of the basis vectors. Unsurprisingly, the dimension of the space spanned by column vectors equals the *rank* of matrix $\mathbf{A}$. One of the fundamental results in linear algebra is that the rank of $\mathbf{A}$ is identical to the dimension of $C(\mathbf{A})$, which in turn is identical to the dimension of $C(\mathbf{A}^\top)$.

C.1.2 Minimizing $\|\mathbf{Ax} - \mathbf{b}\|_2^2$

Let us now look again at the solutions to $\mathbf{Ax} = \mathbf{b}$. In general, there are three different outcomes:

1. there are no solutions to the system
2. there is a unique solution to the system, and
3. there are infinitely many solutions.

[1]As a reminder, two vectors are independent if their linear combination is only zero when both x_1 and x_2 are zero.

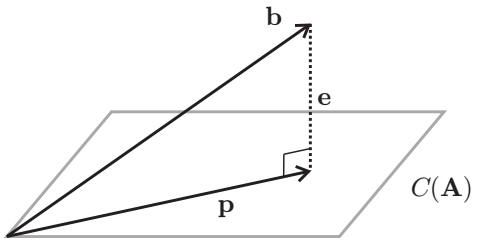


Figure C.1: Illustration of the projection of vector $\mathbf{b}$ *to the column space of matrix* $\mathbf{A}$*. Vectors* $\mathbf{p}$ *(*$\mathbf{b}_c$*) and* $\mathbf{e}$ *(*$\mathbf{b}_l$*) represent the projection point and the error, respectively.*

These outcomes depend on the relationship between the rank (r) of $\mathbf{A}$ and dimensions m and n. We already know that when $r = m = n$ (square, invertible, full rank matrix $\mathbf{A}$) there is a unique solution to the system, but let us investigate other situations. Generally, when $r = n < m$ (full column rank), the system has either one solution or no solutions, as we will see momentarily. When $r = m < n$ (full row rank), the system has infinitely many solutions. Finally, in cases when $r < m$ and $r < n$, there are either no solutions or there are infinitely many solutions. Because $\mathbf{Ax} = \mathbf{b}$ may not be solvable, we generalize solving $\mathbf{Ax} = \mathbf{b}$ to minimizing $\|\mathbf{Ax} - \mathbf{b}\|_2$. In such a way, all situations can be considered in a unified framework.

Let us consider the following example

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix},$$

which illustrates an instance where we are unlikely to have a solution to $\mathbf{Ax} = \mathbf{b}$, unless there is some constraint on b_1, b_2, and b_3; here, the constraint is $b_3 = 2b_2 - b_1$. In this situation, $C(\mathbf{A})$ is a 2D plane in $\mathbb{R}^3$ spanned by the column vectors $\mathbf{a}_1 = (1, 1, 1)$ and $\mathbf{a}_2 = (2, 3, 4)$. If the constraint on the elements of $\mathbf{b}$ is not satisfied, our goal is to try to find a point in $C(\mathbf{A})$ that is closest to $\mathbf{b}$**.** This happens to be the point where $\mathbf{b}$ is projected to $C(\mathbf{A})$, as shown in Figure C.1. We will refer to the projection of $\mathbf{b}$ to $C(\mathbf{A})$ as $\mathbf{p}$. Now, using the standard algebraic notation, we have the following equations

$$\mathbf{b} = \mathbf{p} + \mathbf{e}$$
$$\mathbf{p} = \mathbf{Ax}$$

Since $\mathbf{p}$ and $\mathbf{e}$ are orthogonal, we know that $\mathbf{p}^\top \mathbf{e} = 0$. Let us now solve for $\mathbf{x}$

$$(\mathbf{Ax})^\top (\mathbf{b} - \mathbf{Ax}) = 0$$
$$\mathbf{x}^\top \mathbf{A}^\top \mathbf{b} - \mathbf{x}^\top \mathbf{A}^\top \mathbf{Ax} = 0$$
$$\mathbf{x}^\top \left(\mathbf{A}^\top \mathbf{b} - \mathbf{A}^\top \mathbf{Ax}\right) = 0$$

and thus

$$\mathbf{x}^* = \left(\mathbf{A}^\top \mathbf{A}\right)^{-1} \mathbf{A}^\top \mathbf{b}.$$

This is exactly the same solution as one that minimized the sum of squared errors and maximized the likelihood. The matrix

$$\mathbf{A}^\dagger = \left(\mathbf{A}^\top \mathbf{A}\right)^{-1} \mathbf{A}^\top$$

is called the Moore-Penrose pseudo-inverse or simply a pseudo-inverse. This is an important matrix because it always exists and is unique, even in situations when the inverse of $\mathbf{A}^\top \mathbf{A}$ does not exist.

This happens when $\mathbf{A}$ has dependent columns (technically, $\mathbf{A}$ and $\mathbf{A}^\top\mathbf{A}$ will have the same nullspace that contains more than just the origin of the coordinate system; thus the rank of $\mathbf{A}^\top\mathbf{A}$ is less than n). Let us for a moment look at the projection vector $\mathbf{p}$. We have

$$\begin{aligned}\mathbf{p} &= \mathbf{A}\mathbf{x} \\ &= \mathbf{A}\left(\mathbf{A}^\top\mathbf{A}\right)^{-1}\mathbf{A}^\top\mathbf{b},\end{aligned}$$

where $\mathbf{A}\left(\mathbf{A}^\top\mathbf{A}\right)^{-1}\mathbf{A}^\top$ is the matrix that projects $\mathbf{b}$ to the column space of $\mathbf{A}$.

While we arrived at the same result as in previous sections, the tools of linear algebra allow us to discuss OLS regression at a deeper level. Let us examine for a moment the existence and multiplicity of solutions to

$$\arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 . \tag{C.2}$$

Clearly, the solution to this problem always exists. However, we shall now see that the solution to this problem is generally not unique and that it depends on the rank of $\mathbf{A}$. Consider $\mathbf{x}$ to be one solution to Eq. (C.2). Recall that $\mathbf{x} = \mathbf{x}_r + \mathbf{x}_n$ and that it is multiplied by $\mathbf{A}$; thus any vector $\mathbf{x} = \mathbf{x}_r + \alpha\mathbf{x}_n$, where $\alpha \in \mathbb{R}$, is also a solution. Observe that $\mathbf{x}_r$ is common to all such solutions; if you cannot see it, assume there exists another vector from the row space and show that it is not possible. If the columns of $\mathbf{A}$ are independent, the solution is unique because the nullspace contains only the origin. Otherwise, there are infinitely many solutions. In such cases, what exactly is the solution found by projecting $\mathbf{b}$ to $C(\mathbf{A})$? Let us look at it:

$$\begin{aligned}\mathbf{x}^* &= \mathbf{A}^\dagger\mathbf{b} \\ &= \left(\mathbf{A}^\top\mathbf{A}\right)^{-1}\mathbf{A}^\top(\mathbf{p}+\mathbf{e}) \\ &= \left(\mathbf{A}^\top\mathbf{A}\right)^{-1}\mathbf{A}^\top\mathbf{p} \\ &= \mathbf{x}_r,\end{aligned}$$

as $\mathbf{p} = \mathbf{A}\mathbf{x}_r$. Given that $\mathbf{x}_r$ is unique, the solution found by the least squares optimization is the one that simultaneously minimizes $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ and $\|\mathbf{x}\|_2$ (observe that $\|\mathbf{x}\|_2$ is minimized because the solution ignores any component from the nullspace). Thus, the OLS regression problem is sometimes referred to as the minimum-norm least-squares problem.

Let us now consider situations where $\mathbf{A}\mathbf{x} = \mathbf{b}$ has infinitely many solutions, i.e. when $\mathbf{b} \in C(\mathbf{A})$. This usually arises when $r \leq m < n$. Here, because $\mathbf{b}$ is already in the column space of $\mathbf{A}$, the only question is what particular solution $\mathbf{x}$ will be found by the minimization procedure. As we have seen above, the outcome of the minimization process is the solution with the minimum L_2 norm $\|\mathbf{x}\|_2$.

To summarize, the goal of the OLS regression problem is to solve $\mathbf{x}\mathbf{w} = \mathbf{y}$, if it is solvable. When $k < n$ this is not a realistic scenario in practice. Thus, we relaxed the requirement and tried to find the point in the column space $C(\mathbf{x})$ that is closest to $\mathbf{y}$. This turned out to be equivalent to minimizing the sum of square errors (or Euclidean distance) between n-dimensional vectors $\mathbf{X}\mathbf{w}$ and $\mathbf{y}$. It also turned out to be equivalent to the maximum likelihood solution presented in Section 5.1. When $n < k$, a usual situation in practice is that there are infinitely many solutions. In these situations, our optimization algorithm will find the one with the minimum L_2 norm.

Appendix D

Details on unsupervised representation approaches using factorization

There are many variants of unsupervised learning algorithms that actually correspond to factorizing the data matrix, which we summarize in in Table D.1. In many cases, they simply correspond to defining an interesting kernel on $\mathbf{X}$, and then factorizing that kernel (i.e., kernel PCA). If the entry is empty, this specifies no regularization and no constraints. For a more complete list, see [18] and [20]). As with the regression setting, we can generalize this Euclidean loss to any convex loss

$$L_x(\mathbf{H}, \mathbf{D}, \mathbf{X}) = \sum_{i=1}^{n} L_x(\mathbf{H}_{i:}\mathbf{D}, \mathbf{X}_{i:})$$

where above we used

$$L_x(\mathbf{H}, \mathbf{D}, \mathbf{X}) = \sum_{i=1}^{n} \|\mathbf{H}_{i:}\mathbf{D} - \mathbf{X}_{i:}\|_2^2 = \|\mathbf{HD} - \mathbf{X}\|_F^2 .$$

Algorithms to learn dictionaries

Our focus remains on prediction, and so we would like to use these representations for supervised (or semi-supervised) learning. We use a two-stage approach, where first the new representation is learned in an unsupervised way and then used with supervised learning algorithms. These two stages could be combined into one step with supervised dictionary learning; see [13] for a discussion about this more advanced approach.

The most common strategy to learn these dictionary models is to do an alternating minimization over the variables. The optimization over $\mathbf{D}$ and $\mathbf{H}$ is not jointly convex; however, it is convex in each variable separately. The strategy is to fix one variable, say $\mathbf{H}$, and descend in the other, say $\mathbf{D}$, and then switch, fixing $\mathbf{D}$ and descending in $\mathbf{H}$. This alternating minimization continues until the convergence. Though this is a nonconvex optimization, there is recent evidence that this procedure actually returns the global minimum (see e.g. [12]). We summarize this procedure in Algorithm 5.

Once the dictionary $\mathbf{D}$ and new representation $\mathbf{H}$ have been learned, we can learn the supervised weights $\mathbf{W} \in \mathbb{R}^{k \times m}$ to obtain $\mathbf{HW} \approx \mathbf{Y}$. This can be done with any of the linear regression or classification approaches we have learned so far.

Finally, we need to know how to use these learned models for out-of-sample prediction (i.e., for new samples). The matrices $\mathbf{D}$ and $\mathbf{W}$ contain all the necessary information to perform out-of-sample prediction, and $\mathbf{H}$ does not need to be stored, because it was the representation specific to the training data. For a new sample $\mathbf{x}_{\text{new}}$, the representation can be obtained using

$$\mathbf{h}_{\text{new}} = \operatorname*{argmin}_{\mathbf{h} \in \mathbb{R}^k} L_x(\mathbf{hD}, \mathbf{x}).$$

With the representation for this sample, we can then predict $f(\mathbf{h}_{\text{new}}\mathbf{W})$.

Algorithm	**Loss and constraints**
CCA $\equiv$ orthonormal PLS	$\left\| \begin{bmatrix} \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \\ \mathbf{Y}(\mathbf{Y}^\top \mathbf{Y})^{-1} \end{bmatrix} - \mathbf{HD} \right\|_F^2$
Isomap	$\|\mathbf{K} - \mathbf{HD}\|_F^2$ $\mathbf{K} = -\frac{1}{2}(\mathbf{I} - \mathbf{e}\mathbf{e}')\mathbf{S}(\mathbf{I} - \mathbf{e}\mathbf{e}')$ with $\mathbf{S}_{i,j} = \|\mathbf{X}_{i:} - \mathbf{X}_{j:}\|$
K-means clustering	$\|\mathbf{X} - \mathbf{HD}\|_F^2$ with $\mathbf{H} \in \{0,1\}^{n \times k}$, $\mathbf{H1} = \mathbf{1}$
K-medians clustering	$\|\mathbf{X} - \mathbf{HD}\|_{1,1}$ with $\mathbf{H} \in \{0,1\}^{n \times k}$, $\mathbf{H1} = \mathbf{1}$
Laplacian eigenmaps $\equiv$ Kernel LPP	$\|\mathbf{K} - \mathbf{HD}\|_F^2$ for $\mathbf{K} = \mathbf{L}^\dagger$
Metric multi-dimensional scaling	$\|\mathbf{K} - \mathbf{HD}\|_F^2$ for isotropic kernel $\mathbf{K}$
Normalized-cut	$\left\|(\mathbf{\Lambda}^{-1}\mathbf{X} - \mathbf{HD})\mathbf{\Lambda}^{1/2}\right\|_F^2$ with $\mathbf{H} \in \{0,1\}^{n \times k}$, $\mathbf{H1} = \mathbf{1}$
Partial least squares	$\|\mathbf{X}\mathbf{Y}' - \mathbf{DH}\|_F^2$
PCA	$\|\mathbf{X} - \mathbf{HD}\|_F^2$
Kernel PCA	$\|\mathbf{K} - \mathbf{HD}\|_F^2$
Ratio cut	$\|\mathbf{K} - \mathbf{HD}\|_F^2$ for $\mathbf{K} = \mathbf{L}^\dagger$

Figure D.1: Unsupervised learning algorithms that correspond to a matrix factorization.

Algorithm 5: Alternating minimization for dictionary learning

Input:

inner dimension k
loss L, where $\mathrm{L}(\mathbf{HD}) = L_x(\mathbf{H}, \mathbf{D}, \mathbf{X})$
R_D, the regularizer on $\mathbf{D}$
R_H, the regularizer on $\mathbf{H}$
the regularization weight λ
convergence tolerance
fixed positive step-sizes η_D, η_H
dataset $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$

Initialization:

$\mathbf{D}, \mathbf{H} \leftarrow$ full-rank random matrices with inner dimension k
prevobj $\leftarrow \infty$

Loop until convergence within tolerance or reach maximum number of iterations:

Update $\mathbf{D}$ using one step of gradient descent
Update $\mathbf{H}$ using one step of gradient descent
currentobj $\leftarrow \mathrm{L}(\mathbf{HD}) + \lambda R_D(\mathbf{D}) + \frac{\lambda}{n} R_H(\mathbf{H})$
If $|\text{currentobj} - \text{prevobj}| <$ tolerance, Then break
prevobj $\leftarrow$ currentobj

Output:

$\mathbf{D}, \mathbf{H}$