Modern Methods in Bayesian Probabilistic Modeling and their Applications

Dissertation
zur Erlangung des Grades
„Doktor der Naturwissenschaften"
am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

Konstantin Bob

geb. in Mainz
Mainz, den 19.5.2022

1. Berichterstatter: Prof. Dr. Andreas Hildebrandt
2. Berichterstatter: Prof. Dr. [redacted]
Tag des Kolloquiums: 20.09.2022

D77

Abstract

A fundamental task in all fields of science is to learn from observations. However, undertaking this is usually hindered in two ways: first, the direct observation of the phenomenon may be challenging or impossible, requiring a model of the phenomenon and a statistical approach to separate desired from undesired information. Second, the number of observations may be small, so the resulting uncertainty must be taken into account.

As a resort, the field of Bayesian modeling provides a canonical framework to perform statistical inference from data and prior knowledge in a way that allows to quantify the uncertainty of the results as well. In this approach, probability distributions are used as carriers of information and transformed accordingly. However, in most cases the required computations can only be performed numerically.

In this work we contribute to Bayesian modeling in several ways: first, we present the paraNUTS algorithm for parallelized inference that is formulated in the map-reduce paradigm and achieves considerable speed-ups without significant loss of inference quality. Next, we present TuringOnline.jl, a software package for inference in online settings, that also achieves speed-ups while retaining inference quality to a high degree. Moreover, we present an application of Bayesian modeling to surface topography analysis that yielded action-guiding findings for the field to ensure reproducible results from future studies. Finally, we contribute to nowcasting of infection numbers with our CorCast system that provides the necessary unified treatment of data and models that is extremely important for practical application. Although targeted at the Sars-CoV-2 pandemic, the system is designed to be adopted to other epidemiological modeling easily.

Additionally, the appendices cover research that was not related to Bayesian modeling: first, a scalable and flexible approach to signal classification in mass spectrometry raw data using locality-sensitive hashing and second, a machine learning approach to a classification task in the field of surface topography analysis.

German abstract

Eine grundlegende Aufgabe in allen Bereichen der Wissenschaft ist es, aus Beobachtungen zu lernen. Diese Aufgabe wird jedoch in der Regel auf zweierlei Weise erschwert: Erstens kann die direkte Beobachtung des Phänomens schwierig oder unmöglich sein, so dass ein Modell des Phänomens und ein statistischer Ansatz erforderlich sind, um erwünschte von unerwünschten Informationen zu trennen. Zweitens kann die Anzahl der Beobachtungen gering sein, sodass die daraus entstehende Unsicherheit berücksichtigt werden muss.

Als Ausweg bietet die Bayes'sche Modellierung eine kanonische Rahmenordnung, um statistische Schlussfolgerungen aus Daten und Vorwissen auf eine Weise zu ziehen, die es erlaubt, auch die Unsicherheit der Ergebnisse zu quantifizieren. Bei diesem Ansatz werden Wahrscheinlichkeitsverteilungen als Träger von Informationen verwendet und entsprechend transformiert. In den meisten Fällen können die erforderlichen Berechnungen jedoch nur numerisch durchgeführt werden.

In dieser Arbeit leisten wir auf verschiedene Weise einen Beitrag zur Bayes'schen Modellierung: Zunächst stellen wir den paraNUTS-Algorithmus für parallelisierte Inferenz vor, der im Map-Reduce-Paradigma formuliert ist und erhebliche Geschwindigkeitssteigerungen ohne signifikanten Verlust an Inferenzqualität erzielt. Als Nächstes stellen wir TuringOnline.jl vor, ein Programmpaket für die Inferenz in Fällen, in denen die Eingabedaten erst im Laufe der Zeit bekannt werden. Hiermit werden ebenfalls Geschwindigkeitssteigerungen erzielt und gleichzeitig die Inferenzqualität in hohem Maße beibehalten. Darüber hinaus stellen wir eine Anwendung der Bayes'schen Modellierung auf die Analyse der Oberflächentopografie vor, die zu handlungsleitenden Erkenntnissen für das Feld geführt hat, um reproduzierbare Ergebnisse zukünftiger Studien zu gewährleisten. Schließlich leisten wir einen Beitrag zur Vorhersage von Infektionszahlen mit unserem CorCast-System, das die notwendige einheitliche Behandlung von Daten und Modellen bietet, die für die praktische Anwendung äußerst wichtig ist. Obwohl das System auf die Sars-CoV-2-Pandemie ausgerichtet ist, lässt es sich problemlos auf andere epidemiologische Modellierungen übertragen.

Darüber hinaus decken die Anhänge Forschungsarbeiten ab, die nicht mit der Bayes'schen Modellierung in Zusammenhang stehen: Erstens ein skalierbarer und flexibler Ansatz zur Signalklassifizierung von Massenspektrometrie-Rohdaten unter Verwendung von ähnlichkeitserhaltenden Streuwertfunktionen und zweitens ein maschineller Lernansatz für eine Klassifizierungsaufgabe im Bereich der Oberflächentopographieanalyse.

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Dr. Andreas Hildebrandt, for teaching me how to navigate the academic world over the years and for providing valuable feedback on this work.

Next, I thank my second reviewer, Professor Dr. [redacted], for his quick and extensive comments and improvements to the joint publications.

Furthermore, I would like to thank my colleagues at the Institute, from whom I was able to learn many things.

I would especially like to thank David Teschner, who shared the ups and downs of daily work with me over all these years and with whom I was fortunate to discuss and implement many ideas.

Another thank you goes to the co-authors from the TraCEr laboratory for the good atmosphere during the collaboration and the trust they have placed in me.

Finally, my deepest thanks go to my wife and my parents, who have always supported me and made it possible for me to complete this work.

Contents overview

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Listings

List of Symbols

Notation

$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
$\mathcal{O}(\cdot)$	$f = \mathcal{O}(g)$ if $\limsup_{x\to\infty} \frac{\lvert f(x)\rvert}{g(x)} < \infty$ for strictly positive functions f, g
$\delta(\cdot)$	Dirac distribution, such that $\int_\Omega \delta(\theta - \theta_0)\, f(\theta)\, \mathrm{d}\theta = f(\theta_0)$
$\Gamma(\cdot)$	Gamma function $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x}\, dx$ for $\Re(z) > 0$
$\chi(\cdot)$	Indicator function $\chi_{\mathcal{S}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{S} \\ 0 & \text{if } x \notin \mathcal{S} \end{cases}$
$\mathrm{logit}(\cdot)$	$\mathrm{logit}(p) = \ln\left(\frac{p}{1-p}\right)$ for $p \in (0, 1)$
$A, B, \ldots, X, \Theta$	Random variables
$a, b, \ldots, x, \theta$	Realizations of a random variable
$\pi_A(a)$	Probability density function for A at a
$P_X(x)$	Probability that $X = x$
$P_{A \mid B}(a \mid b)$	Conditional probability that $A = a$ given $B = b$

Probability distributions

$\mathcal{D}$	Generic distribution
$\mathcal{U}(\mathcal{S})$	Uniform distribution over the Borel set $\mathcal{S}$
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
$\mathcal{RN}_{[l,u]}(\mu, \sigma^2)$	Truncated normal distribution with location parameter μ and scale parameter σ on restricted support $[l, u]$
$\mathcal{HN}(\sigma)$	Half-normal distribution with scale σ

$\mathcal{LN}(\mu, \sigma)$	Lognormal distribution with location parameter μ and scale parameter σ
$\mathcal{C}(x_0, \gamma)$	Cauchy distribution with location parameter x_0 and scale parameter γ
$\mathcal{T}(\mu, \sigma, \nu)$	Student-t distribution with probability density $\pi(x\|\mu, \sigma, \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\,\Gamma(\frac{\nu}{2})}\left(1 + \frac{1}{\nu}\left(\frac{x-\mu}{\sigma}\right)^2\right)^{-\frac{\nu+1}{2}}$
$\mathcal{B}(n, p)$	Binomial distribution with n trials and success chance p
$\mathcal{E}(\lambda)$	Exponential distribution with scale λ
$\mathfrak{G}(\mu, \beta)$	Gumbel distribution with location μ and scale β
$\mathcal{NB}(r, p)$	Negative binomial distribution with r failures and success chance p
$\mathcal{P}(\lambda)$	Poisson distribution with mean λ
$\mathfrak{B}(\alpha, \beta)$	Beta distribution with parameters α and β
$\mathcal{G}(\alpha, \beta)$	Gamma distribution with parameters α and β
$\mathcal{IG}(\alpha, \beta)$	Inverse-gamma distribution with shape parameter α and scale parameter b

1. Introduction

Motivation

Learning from data

A fundamental task in all fields of science is to learn from observations. In an idealization, one would be able to study the phenomenon of interest without any interference from other phenomena and one would be able to accumulate as many observations as desired to provide strong evidence for the findings.

Challenges

However, often one encounters circumstances in which neither of those two conditions hold. First, the direct observation of the phenomenon may be challenging or impossible, forcing us to use a model of the phenomenon and a statistical approach to separate desired from undesired information. Second, the number of observations may be small, leaving us uncertain how strongly we belief in our findings. Additionally, often we do not deal with completely new phenomena, but ones for which we have some knowledge from previous research. Leveraging this prior knowledge in a transparent way can be a challenge as well.

Bayesian modeling

Luckily, the field of Bayesian modeling provides a canonical method to perform statistical inference from data and prior knowledge in a way that allows to quantify the uncertainty of the results as well. In this approach, probability distributions are used as carriers of information and transformed accordingly. Similar to the situation in theoretical physics, where the fundamentals like the equations of motion in classical mechanics are known, but the analytical computation of the trajectory can be very complicated, the challenge in Bayesian modeling is that in most cases the solutions can only be found numerically.

Relation to computer science

Thus, Bayesian modeling is related to computer science in several ways: first, Bayesian inference is a concept for processing of information, a core subject of computer science, well reflected in the German translation *Informatik*. Furthermore, the concrete task of numerically solving problems naturally arises in the name computer science as well. Finally, for applications we need efficient algorithms and of course data, whose storage and processing is a challenge of its own, concerning again core concepts in computer science.

Structure

The remainder of this work is structured as follows: we begin with a review of the Bayesian method in chapter 2, discussing major benefits and challenges. Then, in

chapter 3 we review various algorithms used for Bayesian inference, preparing the ground for our own algorithm presented in chapter 4. The following chapter 5 reviews commonly used software libraries for Bayesian modeling, leading over to chapter 6, in which our own contribution to a software package is presented. Next, in chapter 7 we present an application to illustrate the approach and show our findings that had great impact for the field of surface topography analysis. Moreover, in chapter 8 we review epidemiological modeling to prepare the presentation of our contribution to nowcasting of infection numbers in chapter 9. Finally, chapter 10 concludes the work with a summary and an outlook.

Appendices A and B cover research that was not related to Bayesian modeling.

Contributions

The contributions in this work are the following: in chapter 2, chapter 3, chapter 5 and chapter 8 we present a review of the respective topic. In chapter 7, chapter 6, chapter 9 and appendices A and B we present our previously published findings. In chapter 4 we present previously unpublished research.

Part I.

Methodology

2. Learning from uncertain information through Bayesian inference

After a general consideration on learning from data by mathematical models, this chapter introduces the framework of Bayesian inference, starting with its axiomatic foundations, the main theorem and benefits for users. We conclude the chapter with a short comparison to other frameworks.

2.1. Models and learning from data

Different domains and steps of statistical inference and prediction

Figure 2.1 shows the different domains and steps in the process of statistical inference and prediction. In general, the starting point is that we have questions q on phenomena in what we will call the 'real world' and we would like to answer those questions. Properly designed experiments, denoted by E in Figure 2.1, allow us to map the question to recorded data X. As the recorded data often has still too much information in it, we try to compress this information by a mathematical [1] model $\mathcal{M}$, that usually has free parameters Θ. Inferring suited parameters from the data is often a core task, requiring the maps denoted by L and I in Figure 2.1, which allow us to compute the degree of agreement between model and data and how much belief we have in possible parameter values, given the data, respectively. Next, models can be used to predict parts of the data domain that were e.g. not measured, creating new data $\tilde{X}$. In order to achieve this, we need a map for prediction, called P in Figure 2.1. Finally, based on model predictions we want to make decisions, represented by the map D in Figure 2.1, that lead to actions a in the 'real world'.

Choosing maps

As the previous description shows, a set of maps E, L, I, P, D is required to perform statistical inference and prediction. As there is a vast complexity of circumstances and problems, the individual choices of the maps E, L, I, P, D may differ wildly and can be very problem specific.

Bayesian framework

However, there are certain frameworks which have developed principles on choosing the maps E, L, I, P, D. Especially, in this chapter we will see how the Bayesian framework deals with the inference map I and the prediction map P and, to a lesser extent, the decision map D.

Model finding and comparison

Finding a good model M and the corresponding map L in Figure 2.1 is often

[1] That indeed models using abstract mathematics are useful for the description of phenomena in the real world is well discussed by Wigner [208].

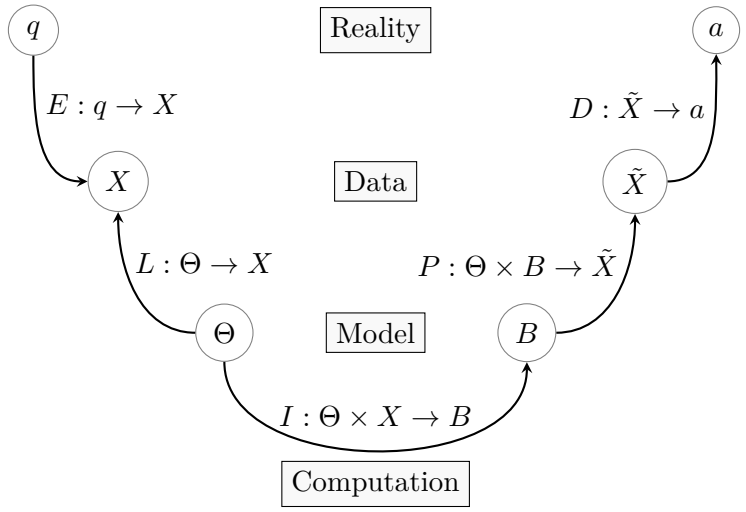


Figure 2.1.: Domains and steps in the process of statistical inference and prediction. In general, statistical inference consists of several steps, that we will consider here as maps between different domains. In the first domain, which we will call briefly *reality* without going into philosophical considerations, questions q and actions a reside, while in the second domain called *data*, observations from the reality are recorded as data X and $\tilde{X}$. In the third domain, called *model*, we locate mathematical models described by parameters Θ and degree of belief B in a parameter. Finally, the domain called *computation* deals with the challenges that arise for actual problems.

perceived as a core task in science and successful models have provided famous scientific breakthroughs, e.g. by Einstein [63] or Schrödinger [180]. Next to agreement with the observed data, other requirements have been formulated for good models like testability and others, see [70] for more details. The more concrete task of model comparison is further discussed in e.g. [54].

Experimental design

As it would otherwise go beyonds the scope of this work, we only refer the reader to the literature about principles on choosing E, i.e. e.g. [69] on the art of experimentation and e.g. [134] on experimental design.

Limits by epistemology

We conclude this section with a word of caution, namely that inductive learning from data is always prone to error, even if perfect maps E, L, I, P, D were available. That is due to fundamental, epistemological reasons. As Popper [164] argues, we can only falsify models, for which a single observation suffices, but even a large number of observations in agreement does not allow us to assume that a model was correct. Taleb [192] provides further elaboration on this, both for science and other fields, while Prado [165] covers problems in learning from possibly non-stationary time-series in greater details.

2.2. Probability theory as an extended logic

Extending Boolean logic

From the discussion above we can clearly see that there is an interest in frameworks that could provide maps like I and P in a principled way. If we were to deal with absolutely true or false propositions, the correct framework would be that of Boolean logic. When, however, reasoning about model parameters Θ, the problem is often that due to limited information there is no absolute certainty, but only varying degrees of it. We will now follow the arumentation of Jaynes [101], who has developed a way to extend the deductive reasoning in Boolean logic to the manipulation of degrees of plausibility, containing Boolean logic as a limiting case.

Axiomatic origins

In his endeavor, Jaynes [101] starts with a three basic desiderata: the first states that plausibilities may be represented by real numbers. The second desideratum describes some requirements on correspondence to Boolean logic. Finally, the third requirement is that of consistency among deduced expressions.[2] From these ideas he develops a complete framework for manipulating degrees of plausibility of combined propositions.

Probability interpretation

In this framework, Jaynes [101] defines probabilities as a measure for the degree of plausibility, thus avoiding the connection to infinite limits of observed event frequencies, rather interpreting it as a subjective quantity that reflects a state of knowledge. However, in order to emphasize that his framework does not depend on particular interpretations, Jaynes [101] shows that it can be considered equivalent to Kolmogorov's axioms. The debate on different interpretation of probability has been going on for long times, see e.g. the work [66] and references therein to Kant and Laplace.

2.3. Bayesian terminology and eponymous theorem

Being assured that we have a solid underlying theory, we can now come to the core of the Bayesian framework, it's inference map I.

Prior

The framework is built upon two cornerstones: the first one is the so-called prior distribution $P_\Theta(\theta)$, which encodes the belief of the modeler on the parameter set Θ, prior to having seen the data X, as a probability distribution. Note that the term prior can in some circumstances be meant in a logical sense, not a timely or causal sense.

Likelihood

The second one is the map L, that describes how well the predicted values of the model $\mathcal{M}$ fit the observed data X. In the Bayesian framework it is called likelihood $P_{X\,|\,\Theta}(x\,|\,\theta)$ and is also considered a conditional probability.

Posterior

The quantity of actual interest is the so-called posterior $P_{\Theta\,|\,X}(\theta\,|\,x)$, that is the conditional probability for the parameter set Θ after having seen the data X.

The connection between the posterior and both prior and likelihood is given by

[2] See first chapter of [101] for details.

the following theorem, that was eponymous for the whole framework:

Theorem 1 (Bayes' theorem)**.** *Given observed data X, a parameterized model $\mathcal{M} = \mathcal{M}(\Theta)$ with prior $P_\Theta(\theta)$ and likelihood $P_{X\,|\,\Theta}(x\,|\,\theta)$, the posterior is given by*

$$P_{\Theta\,|\,X}(\theta\,|\,x) = \frac{P_{X\,|\,\Theta}(x\,|\,\theta)}{P_X(x)}\, P_\Theta(\theta), \tag{2.1}$$

where $P_X(x)$ denotes the probability of the observed data and is called evidence.

Proof

While to some, equation (2.1) is a simple fact concerning conditional probabilities, Jaynes [101] gives a proof that shows the deeper origins of the equation and comments on the philosophical consequences and interpretations. The first occurrence of the results, albeit in other form, dates back to the work of Bayes [11].

Updating belief

Among the several interpretations[3], we want to emphasize the following: using Bayes' theorem can be seen as updating ones knowledge on arrival of previously unseen data, using the likelihood as a kind of filter to rule out propositions with low plausibility. Sticking to the exact form of Bayes' theorem ensures that we reason in accordance to the desiderata of Jaynes [101] or more technically phrased that we have a closed mapping from the set of states of knowledge to itself.

Normalization

Before we can use the theorem for inference in general, we have to eliminate the evidence $P_X(x)$ from (2.1). In order to do so, we can exploit the fact that the posterior needs to be normalized. Consider for simplicity the case of a continuous parameter Θ with domain Ω.[4] Then we can compute

$$\begin{aligned} 1 &= \int_\Omega P_{\Theta\,|\,X}(\theta\,|\,x)\,\mathrm{d}\theta \\ &= \int_\Omega \frac{P_{X\,|\,\Theta}(x\,|\,\theta)\, P_\Theta(\theta)}{P_X(x)}\mathrm{d}\theta \\ &= \frac{\int_\Omega P_{X\,|\,\Theta}(x\,|\,\theta)\, P_\Theta(\theta)\,\mathrm{d}\theta}{P_X(x)}, \end{aligned}$$

as the evidence $P_X(x)$ does not depend on the parameter set Θ. Substituting back

[3] See e.g. [126] for one using differential geometry.

[4] The general case of a full measure-theoretic treatment of Bayes' theorem, is given by Schervish [178, theorem 1.31], where one considers a parameter Θ from a probability space $(\Omega, \mathcal{F}, \mu_\Theta)$, where Ω denotes the respective sample space, $\mathcal{F}$ the σ-algebra and μ_Θ the probability measure. If some technical conditions hold, we can identify μ_Θ with the prior distribution and the posterior with the Radon-Nikodym derivative

$$\frac{\mathrm{d}\mu_{\Theta\,|\,X}}{\mathrm{d}\mu_\Theta}(\theta\,|\,x) = \frac{\pi_{X\,|\,\Theta}(x\,|\,\theta)}{\int_\Omega \pi_{X\,|\,\Theta}(x\,|\,\theta)\,\mathrm{d}\mu_\Theta(\theta)}, \tag{2.2}$$

where $\pi_{X\,|\,\Theta}$ denotes the conditional density of X, which we identify with the likelihood. As the usage of a Radon-Nikodym derivative indicates technically the change of a measure, using Bayes' theorem can be readily interpreted as a change in how we distribute our belief after new information arrived.

into equation (2.1), we find

$$P_{\Theta \mid X}(\theta \mid x) = \frac{P_{X \mid \Theta}(x \mid \theta)\, P_{\Theta}(\theta)}{\int_{\Omega} P_{X \mid \Theta}(x \mid \theta)\, P_{\Theta}(\theta)\, \mathrm{d}\theta}. \tag{2.3}$$

Universal formulation

With this result, we are in principle ready to start with inference problems, as equation (2.3) is a universal formulation that tells us how to combine prior and likelihood to the posterior. Thus, one could compare equation (2.3) to Newton's laws or the Lagrangian or Hamiltonian formulation of classical mechanics that govern solving problems in their field.

The next chapter will show that in accordance to what the comparison to classical mechanics suggests, the real difficulty lies in the actual computation of the integral in equation (2.3), much like when solving equations of motion in classical mechanics. In both cases, the operations involved are not closed on the set of simple and well-behaved functions, potentially mapping those to ones which cannot be represented by analytical expressions.

However, before turning to concrete problems, we want to characterize the framework and its consequences for application on a more abstract level.

2.4. Characteristics of Bayesian inference

Prior knowledge

Usage of prior distributions $P_{\Theta}(\theta)$ is one of the distinct aspects of Bayesian inference. It offers the advantage to leverage expert knowledge in a quantitative and transparent way, which however sometimes is criticized as being subjective or as a way of manipulating the outcome of a inference. While for large sample sizes or when the likelihood is overwhelmingly restrictive, the influence of the prior on the estimates vanishes, in other cases there are ways to quantify the prior influence as the size of a hypothetical sample that would have the same effect on the posterior, see [136] [145] [207] for details. Note that in other frameworks there are common practices that act like priors, see section 2.5 for details.

Corner cases and implicit assumptions

A less obvious characteristic of Bayesian inference is that it does not require implicit assumptions on the problem, such that no ill-defined corner cases can occur. Jaynes [101] gives many examples.

Uncertainty quantification and propagation

By using probability distributions as descriptions of a state of knowledge we are able to quantify uncertainty in our estimates, by measures like standard deviations or as Jaynes [101] suggests by the entropy of a distribution. Furthermore, by using Bayes' theorem the uncertainty can be propagated during inference. We will now see two situations, in which the Bayesian framework manages to take the uncertainty of inference into account by use of marginalization.

Prediction by model averaging

The map P in Figure 2.1 for making predictions on new data $\tilde{X}$ that the Bayesian framework suggests, is the use of *posterior predictive distributions*

$$P_{\tilde{X} \mid X}(\tilde{x} \mid x) = \int_{\Omega} P_{\tilde{X} \mid \Theta}(\tilde{x} \mid \theta)\, P_{\Theta \mid X}(\theta \mid x)\, \mathrm{d}\theta, \tag{2.4}$$

that is by marginalization over the parameter set Θ. (2.4) can be interpreted as form of model averaging, that is each possible generating model $P_{\tilde{X}\,|\,\Theta}(\tilde{x}\,|\,\theta)$ is weighted by its plausibility $P_{\Theta\,|\,X}(\theta\,|\,x)$.

In order to assess the modeling choices, it is sometimes useful to consider a related concept, the *prior predictive distribution*

$$\int_{\Omega} P_{\tilde{X}\,|\,\Theta}(\tilde{x}\,|\,\theta)\, P_{\Theta}(\theta)\,\mathrm{d}\theta, \tag{2.5}$$

where the prior is used instead of the posterior for averaging. When comparing equation (2.5) to equations (2.1) and (2.3) one finds that the prior predictive distribution equals the evidence $P_X(x)$, but offers a different interpretation of the term.

Elimination of nuisance parameters

If the parameter set Θ of our model consists of several random variables, sometimes we are not interested in some of them, which are called nuisance parameters accordingly. Often they are however necessary to make the model complete. Consider now the case that $\Theta = (\Phi, \Lambda)$, where Φ is the set of parameters that we are actually interested in and Λ the set of nuisance parameters. By marginalization we find

$$P_{\Phi\,|\,X}(\phi\,|\,x) = \int_{\Omega_\Lambda} P_{\Phi,\Lambda\,|\,X}(\phi, \lambda\,|\,x)\,\mathrm{d}\lambda, \tag{2.6}$$

where Ω_Λ denotes the sample space of Λ. Applying Bayes' theorem (2.1) and the definition of conditional probability, we find

$$\begin{aligned} P_{\Phi\,|\,X}(\phi\,|\,x) &= \int_{\Omega_\Lambda} \frac{P_{X\,|\,\Phi,\Lambda}(x\,|\,\phi,\lambda)\, P_{\Phi,\Lambda}(\phi,\lambda)}{P_X(x)}\mathrm{d}\lambda && (2.7)\\ &= \int_{\Omega_\Lambda} \frac{P_{X\,|\,\Phi,\Lambda}(x\,|\,\phi,\lambda)\, P_{\Lambda\,|\,\Phi}(\lambda\,|\,\phi)\, P_{\Phi}(\phi)}{P_X(x)}\mathrm{d}\lambda && (2.8)\\ &= \frac{P_{\Phi}(\phi)}{P_X(x)} \int_{\Omega_\Lambda} P_{X\,|\,\Phi,\Lambda}(x\,|\,\phi,\lambda)\, P_{\Lambda\,|\,\Phi}(\lambda\,|\,\phi)\,\mathrm{d}\lambda. && (2.9) \end{aligned}$$

By this result we see that elimination of nuisance parameters can be handled by adapting the likelihood to the so-called marginal likelihood, leaving the rest of the framework unchanged. This straightforward treatment of nuisance parameters is in large contrast to other frameworks, that have considerable fundamental problems when dealing with nuisance parameters, see [9] for details.

Decisions under uncertainty

Also for the map D in Figure 2.1, the Bayesian framework offers a solution: given a set of decisions $\mathcal{D} = \{d_i\}_{i=1}^n$ that depend on a inferred model parameter Θ and a utility function $u : \mathcal{D} \times \Omega_\Theta \to \mathbb{R}$ that expresses the utility of making a decision under the estimated parameter. Jaynes [101] then gives the following rule:

$$\hat{d} = \arg\max_{d\in\mathcal{D}} \int_{\Omega_\Theta} u(d,\theta)\, P_{\Theta\,|\,X}(\theta\,|\,x)\,\mathrm{d}\theta, \tag{2.10}$$

that is to make the decision that maximizes the posterior-averaged utility. Again we see that it is advantageous to compute full probability distributions, as the uncertainty from the inference gets taken into account as well.

2.5. Comparison to optimization based methods

Maximum likelihood

As shown in the previous section, the Bayesian framework is a universal formulation of reasoning. Thus, it also contains other statistical frameworks as special cases. One common way of doing parameter inference is the *maximum likelihood principle*,

$$\hat{\theta} = \arg\max_{\theta \in \Omega} P_{X \mid \Theta}(x \mid \theta), \tag{2.11}$$

where Ω denotes the sample space of Θ. In the field of machine learning, often instead the function

$$l = -\ln P_{X \mid \Theta}(x \mid \theta), \tag{2.12}$$

called *loss*, is minimized.

Bayesian equivalent

In the Bayesian framework, this is equivalent to using a prior that is constant or approximately constant around the maximum of the posterior, such that the posterior is dominated by the likelihood and using the posterior argument maximum as the best parameter estimate, such that one could write

$$P_{\Theta \mid X}(\theta \mid x) = \delta(\theta - \hat{\theta}), \tag{2.13}$$

where δ denotes the Dirac distribution. Consequently, by using the best estimate only, the predictive distributions (2.4) of those approaches are given by

$$P_{\tilde{X} \mid X}(\tilde{x} \mid x) = \int_{\Omega} P_{\tilde{X} \mid \Theta}(\tilde{x} \mid \theta)\, \delta(\theta - \hat{\theta}) \mathrm{d}\theta = P_{\tilde{X} \mid \Theta}(\tilde{x} \mid \hat{\theta}), \tag{2.14}$$

which could lead to strong underestimation of the uncertainty involved.

Regularization as a prior

Interestingly, sometimes in other frameworks a technique called regularization is used, where a term $r(\theta)$ is added to the loss, such that

$$l_r = -\ln(P_{X \mid \Theta}(x \mid \theta)) + r(\theta), \tag{2.15}$$

in order to prevent overfitting. When comparing with Bayes' theorem (2.1), we find that this is equivalent to a prior $P_{\Theta}(\theta) = \exp(-r(\theta))$. A more subtle use of what in the Bayesian framework is called prior information in other frameworks is the use of starting values in numerical optimization, which is often not incorporated in the model description, but mentioned somewhere else, maybe with a remark that this helps against numerical instability.

Computational constraints

While theoretical considerations are in favor of Bayesian inference, in practice sometimes one cannot afford to use the Bayesian framework due to computational restrictions. chapter 3 will cover that in detail.

2.6. Modeling workflow

Gelman et al. [75] argue that in a real world setting, there is more work needed than the mere posterior inference, calling it workflow. Figure 2.2 shows a flowchart of the concept. Examples include checking that priors are sensible e.g. using the prior predictive distribution (2.5) and checking the fit of the models on data level. Also the different steps shown in Figure 2.1 require steps on top of the posterior inference, partially overlapping with those shown in Figure 2.2. During the remainder of this work, we will come back to points addressed in Figure 2.2 at the corresponding opportunities.

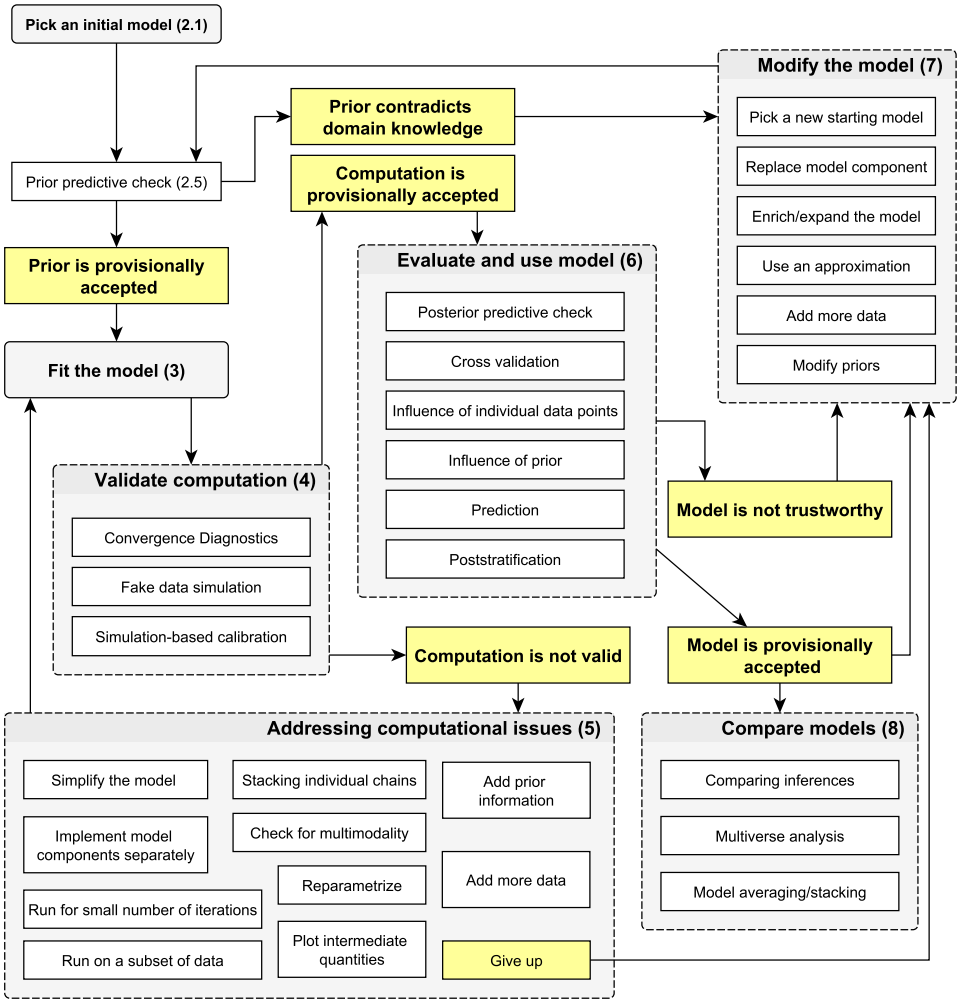


Figure 2.2.: Bayesian workflow.
The flowchart shows steps required in a Bayesian workflow. Note that posterior inference is only a small part of the complete picture. The numbering of the boxes refers to the sections of the work [75], from which the image was taken.

3. Algorithms for Bayesian inference

As it was shown in chapter 2, Bayes' theorem (2.3) provides a universal formulation for inference. However, when applying it to practical problems, we are forced to handle integrals of the form

$$P_X(x) = \int_\Omega P_{X\,|\,\Theta}(x\,|\,\theta)\, P_\Theta(\theta)\,\mathrm{d}\theta, \tag{3.1}$$

for a continuous[1] parameter Θ with domain Ω.

In this chapter we will discuss different approaches, both on an abstract and concrete level, using two example cases.

3.1. Analytical computations

Under certain circumstances it is possible to tract a Bayesian inference completely analytically by solving the integral in (3.1). This represents the ideal case, which does however happen only in rare cases.

Conjugate priors

One technique to facilitate the derivation of closed-form solutions is the usage of so-called conjugate priors, which are prior distributions that come from the same family of distributions as the likelihood does. For a list of matching pairs of likelihood and prior see e.g. [67].

Example case: Beta-binomial model

We will use the beta-binomial model as an typical example for a case that can be solved in closed form, see textbooks by Gelman et al. [74] and Kruschke [109]. The example case will also be illustrating for some theoretically discussed aspects from the last chapter.

Experiment and observed data

Following the scheme in Figure 2.1, we consider an experiment E which consists of n independent trials with a binary outcome, often called 'failures' and 'successes' and a fixed success probability Θ. Moreover, by X we denote the number of successes. To keep the discussion general, we do not want to fix a setting, as the model can be used in a variety of circumstances. A recent case with large public interest was the efficacy analysis of a newly developed, mRNA-based Covid-19 vaccine by Polack et al. [163].

[1] There is no loss of generality when assuming a continuous parameter. See footnote around equation (2.2) for the general case that involves evaluation of a measure over the whole space.

Model

As a next step, we build a model $\mathcal{M}$ by assuming identical probability for success and independence of the trials, such that outcome can be described by a binomial distribution. In order to allow for a closed form solution, we choose the conjugate prior to the binomial distribution, a beta distribution. Taken together the model is given by

$$\Theta \sim \mathfrak{B}(\alpha, \beta) \tag{3.2}$$
$$X \sim \mathcal{B}(n, \Theta). \tag{3.3}$$

In order to compute the posterior analytically, we inspect the densities:

Beta prior

First, equation (3.2) translates to

$$\pi_\Theta(\theta) = \frac{\theta^{\alpha-1}\,(1-\theta)^{\beta-1}}{B(\alpha,\beta)}, \tag{3.4}$$

where $B(\alpha, \beta)$ denotes the Beta function that is defined as

$$B(\alpha,\beta) = \int_0^1 \tau^{\alpha-1}(1-\tau)^{\beta-1}\,d\tau \tag{3.5}$$

for complex numbers α, β with strictly positive real part. Alternatively, it can be written as $B(\alpha,\beta) = \frac{\Gamma(\alpha)\,\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ in terms of the gamma function $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}\,dx$ for $\Re(z) > 0$.

Binomial likelihood

Second, equation (3.3) translates to

$$P_{X\,|\,\Theta}(x\,|\,\theta) = \binom{n}{x}\,\theta^x\,(1-\theta)^{n-x}. \tag{3.6}$$

Inference

When closely observing the structure of (3.4) and (3.6), it becomes clear that the product of $\pi_\Theta(\theta)$ and $P_{X\,|\,\Theta}(x\,|\,\theta)$ will have the structure $\ldots\theta^{\cdots}(1-\theta)^{\cdots}$, thus being proportional to a beta distribution again. Using equation (3.5) we find that

$$P_X(x) = \frac{\binom{n}{x}}{B(\alpha,\beta)}\int_0^1 \theta^{\alpha-1+x}\,(1-\theta)^{\beta-1+n-x}\,\mathrm{d}\theta = \frac{\binom{n}{x}}{B(\alpha,\beta)}B(\alpha+x,\beta+n-x), \tag{3.7}$$

yielding the posterior density

$$\pi_{\Theta\,|\,X}(\theta\,|\,x) = \frac{\theta^{\alpha-1+x}\,(1-\theta)^{\beta-1+n-x}}{B(\alpha+x,\beta+n-x)} \tag{3.8}$$

and indicating that

$$\Theta|X \sim \mathfrak{B}(\alpha+X, \beta+n-X), \tag{3.9}$$

such that the posterior is again a Beta distribution.

Finally, we calculate the posterior predictive distribution for observing $\tilde{X} = \tilde{x}$ in $\tilde{n}$ trails, Predictive distribution

$$P_{\tilde{X}\,|\,X}(\tilde{x}\,|\,x) = \int_0^1 \binom{\tilde{n}}{\tilde{x}} \theta^{\tilde{x}}\,(1-\theta)^{\tilde{n}-\tilde{x}}\,\frac{\theta^{\alpha-1+x}\,(1-\theta)^{\beta-1+n-x}}{B(\alpha+x,\beta+n-x)}\,\mathrm{d}\theta \tag{3.10}$$

$$= \binom{\tilde{n}}{\tilde{x}} \frac{B(\alpha+x+\tilde{x},\beta+n-x+\tilde{n}-\tilde{x})}{B(\alpha+x,\beta+n-x)}, \tag{3.11}$$

using again the structural similarity to the beta distribution.

By comparing (3.8) and (3.4) one finds that the parameter α of $\pi_\Theta(\theta)$ corresponds to x in $P_{X\,|\,\Theta}(x\,|\,\theta)$ and the parameter β of $\pi_\Theta(\theta)$ to $n-x$ in $P_{X\,|\,\Theta}(x\,|\,\theta)$, by which one concludes that the effective sample size of the beta distribution prior is $n_{\text{eff}} = \alpha + \beta$, see [136] for details. Consequently, we can compare $n_{\text{eff}} = \alpha + \beta$ to n when reasoning about the prior strength. Prior influence

For comparison we compute the maximum likelihood estimator by Maximum likelihood estimate

$$0 = \frac{\partial}{\partial\theta}\,P_{X\,|\,\Theta}(x\,|\,\theta) = \binom{n}{x}\,\theta^{x-1}\,(1-\theta)^{n-x-1}(x-\theta\,n), \tag{3.12}$$

which yields[2]

$$\hat{\theta} = \frac{x}{n}, \tag{3.13}$$

allowing for an interpretation as a frequency. The resulting predictive distribution is $p_{\tilde{X}\,|\,\Theta}(\tilde{x}\,|\,\hat{\theta})$, here a binomial distribution.

Figure 3.1 shows plots of the distributions for the example case $x = 7$ and $n = 10$. Figure 3.1a shows two choices of prior distributions, a flat one with $n_{\text{eff}} = 2$, shown by blue linecolor, and a more informative one with $n_{\text{eff}} = 10$ centered at $\theta = 0.5$, shown by orange linecolor. Discussion on example data

The resulting posterior distributions are shown in Figure 3.1b in corresponding line colors as well as the maximum likelihood estimator in black linecolor. As described in the previous chapter, the mode of the posterior resulting from the flat prior matches the maximum likelihood estimator. In contrast to that, the posterior resulting from the more informative prior has its mode at $\theta = 0.6$, being in line with the intuition that due to prior effective sample size n_{eff} and sample size are the same, the mode equals the mean estimate of those two 'information sources'.

Finally, Figure 3.1c shows the resulting posterior predictive distributions for $\tilde{n} = 10$. When comparing the posterior predictive distribution based on the flat prior, shown by blue linecolor, with the predictive distribution by the maximum likelihood approach, shown by black linecolor, it becomes evident that the posterior predictive distribution is broader due to taking the uncertainty of the estimate into

[2] Checking by $\frac{\partial^2 P_{X\,|\,\Theta}(x\,|\,\theta)}{\partial\theta^2}|_{\theta=\hat{\theta}} = -n\binom{n}{x}\left(\frac{x}{n}\right)^{x-1}\left(1-\frac{x}{n}\right)^{n-x-1} < 0$ that $\hat{\theta}$ is indeed a maximum.

account as well, which is line with the discussion from the previous chapter. The posterior predictive distribution based on the more informative prior, shown by orange linecolor, is again centered differently as already the posteriors were.

3.2. Distributional approximation

An alternate method for Bayesian inference is distributional approximation, where one assumes a certain mathematical form of the posterior distribution that is described by hyperparameters. Often this is used for models with very many parameters like large neural networks, where sampling is computationally too expensive and the interactions of parameters is not of particular interest, giving rise to the so-called mean-field posterior, i.e. products of normal distributions.

Inference is then usually achieved by solving an optimization problem like finding the parameter set with minimum Kullback-Leiber divergence to a target function, see [74] for details.

3.3. Numerical quadrature

When a closed form solution of (3.1) is not available, there is the possibility to solve (3.1) approximately by numerical quadrature, which in the general case means to use

$$\int_\Omega f(\xi)\,\mathrm{d}\mu(\xi) \approx Q_n(f) = \sum_{i=1}^{n} w_i\, f(\xi_i), \tag{3.14}$$

where w_i are called weights and ξ_i nodes. Choice of w_i and ξ_i has a strong influence on the approximation error

$$\Delta = \left| \int_\Omega f \mathrm{d}\mu(\xi) - Q_n(f) \right|, \tag{3.15}$$

see e.g. for [84] different variants and [167] for aspects of implementation.

Curse of dimensionality

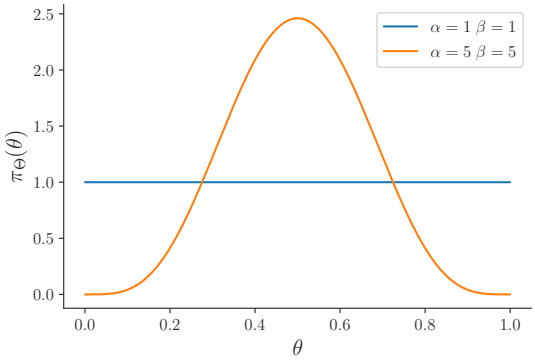
While this may be suitable if Ω has only a few dimensions, it does not work well if Ω is high-dimensionsal, as Müller-Gronbach, Novak, and Ritter [137] show: Given a maximum allowed error ϵ such that $\Delta < \epsilon$, the required number n of nodes ξ_i grows exponentially with $d = \dim(\Omega)$,

$$n = \mathcal{O}(\epsilon^d), \tag{3.16}$$

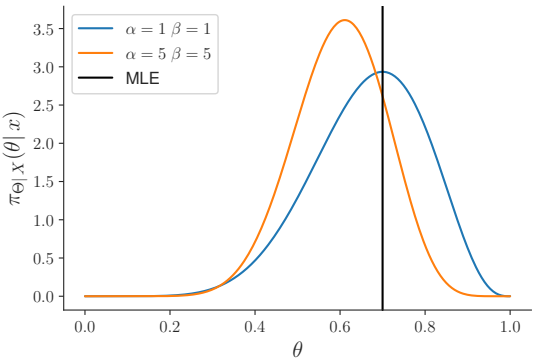
for a large class of functions f and deterministic choices of w_i and ξ_i, see section 7.4.2 in [137] for details. This phenomenon is known as the *curse of dimensionality*.

This, together with a further problem that we will encounter in the next section, leads to the fact that numerical quadrature is a rather uncommon way of computing (3.1) for Bayesian inference.

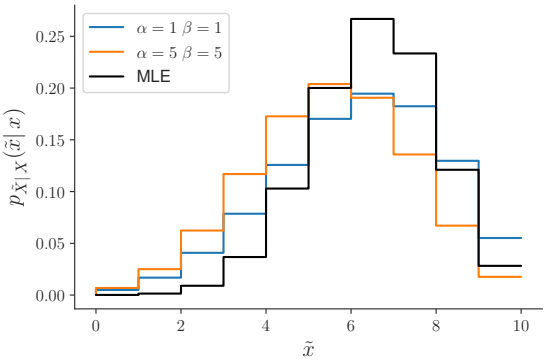
McElreath [127] discusses grid based sampling approximations for which the same arguments as for numerical quadrature apply.



(a) Prior density distributions



(b) Posterior density distributions



(c) Posterior predictive distributions

Figure 3.1.: Prior, posterior and posterior predictive distribution for a beta-binomial model on example data $x = 7$, $n = 10$ and $\tilde{n} = 10$. Two different priors were used and contrasted to the results from maximum likelihood estimation (MLE). See text in section 3.1 for more details.

3.4. Random sampling or Monte Carlo techniques

As shown in the previous section, deterministic methods for numerical quadrature suffer from the curse of dimensionality. In this section we will see that randomized algorithms can circumvent that problem, of course at the cost of other difficulties, whose solution we look at in turn.

Many of these methods have Monte Carlo in their name, which is a reference to the famous casino and goes back to studies of problems in physics, see the article by Metropolis and Ulam [130] for details. A detailed survey on Monte Carlo methods with a focus on parameter estimation is given by Luengo et al. [119].

3.4.1. Monte Carlo integration

Interestingly, but maybe counter intuitively, randomized algorithms are better suited for high-dimensional quadrature problems in the following sense: consider again the general quadrature scheme (3.14)

$$Q_n(f) = \sum_{i=1}^{n} w_i \, f(\xi_i), \tag{3.17}$$

but now with the choices

$$w_i = \frac{1}{n} \tag{3.18}$$

and ξ_i as realizations of the random variable

$$\Xi \sim U(\Omega) \tag{3.19}$$

which is following a uniform distribution $U(\Omega)$.

Asymptotic behavior

Then $Q_n(f)$ converges in a stochastic sense to the integral of f and the error of approximation is given by

$$\Delta = \frac{1}{\sqrt{n}} \, \sigma(\Xi), \tag{3.20}$$

where $\sigma(\Xi)$ denotes the standard deviation of Ξ, see [137, Section 3.1] for details. Furthermore, it can be shown [137, Exercise 7.7] that for a large class of functions f

$$\Delta \leq \sqrt{\frac{d}{12n}}, \tag{3.21}$$

where $d = \dim(\Omega)$. Thus the required number n of nodes ξ_i such that $\Delta < \epsilon$ grows only weakly with d compared to the case in equation (3.16). For more details see Müller-Gronbach, Novak, and Ritter [137, Chapter 7], who give an extended discussion.

Taken together, Monte Carlo integration allows in principle to compute integrals

of high-dimensional integrands without suffering from the curse of dimensionality. There are however other problems that arise:

Random number generation

For practical use, realizations of the random variable Ξ are required, which is a non-trivial task when one recalls that computers are complicated, but still deterministic machines. Typically, either measurements of physical process that are considered random or so-called pseudo-random generators are used. While the first source of random variables is more attractive from a theoretical point of view, it cannot be instantiated arbitrarily often, the second source requires careful analysis of the results before it can be considered trustworthy, see e.g. the discussion by Neumann [146] or in [167] and [137][3]. As a result, the Mersenne twister by Matsumoto and Nishimura [125] with subsequent versions and a generator by Panneton, L'Ecuyer, and Matsumoto [154] were developed to solve the respective issues.

Improving practical performance

Although we have seen that asymptotically Monte Carlo integration yields the correct results, for practical purposes, we are only able to approximate the result by a finite number of steps. As the error in (3.20) is proportional to $\sigma(\Xi)$, so-called variance reduction techniques were developed, see e.g. chapter 5 in [137] for several examples.

Importance sampling

One technique of particular interest is the so-called *importance sampling*, which replaces the uniform distribution $U(\Omega)$ in Ω by a distribution D that puts more emphasis on regions where f has large values and thus drastically improves efficiency. However, in practice, it can be hard to specify or sample from D, see chapter 6 in [137] for more details.

3.4.2. The Markov chain Monte Carlo method

While the previous approaches tried to compute or estimate the integral in (3.1), the Markov chain Monte Carlo approach circumvents this by exploiting the structure of Bayes' theorem (2.1). Before we can discuss the approach in detail, we need to introduce some concepts at first, following chapter 6 in [137].

Markov chains

An indexed set of random variables $\{\Lambda_i\}_{i\in I}$ which are all defined in one probability space $(\Omega, \mathcal{F}, \mu)$ and take values in one measurable state space S, is called a stochastic process [137, p. 96].

Markov chains are stochastic processes with two further defining properties: first, the index i is considered discrete, i.e. e.g. $I \subseteq \mathbb{N}$ and second, the so-called Markov property:

$$P_{\Lambda_i|\{\Lambda_j\}_{j=1}^{i-1}}(\lambda_i|(\lambda_1,\ldots,\lambda_{i-1})) = P_{\Lambda_i|\Lambda_i-1}(\lambda_i|\lambda_{i-1}), \tag{3.22}$$

which is often called 'memorylessness'. Markov chains are often used as models in different fields like machine learning [167], mathematical finance [97], bioninformatics [59] or for internet search engines [152].

[3] Even more complicated is the simulation of stochastic processes in continuous time, see e.g. the discussion by Knuth [106].

Stationary distribution

If we further assume that the state space S has a finite dimension $d = \dim(S) < \infty$, we can represent the states as vectors $\boldsymbol{\lambda}$ and the transition probabilities (3.22) as a matrix $\mathbf{P}$. The state $\underline{\boldsymbol{\lambda}}$ for which

$$\underline{\boldsymbol{\lambda}} = \mathbf{P}\,\underline{\boldsymbol{\lambda}} \tag{3.23}$$

is called *stationary state* or *stationary distribution* and will be of particular interest. Existence and well-definedness of $\underline{\boldsymbol{\lambda}}$ depends of course on $\mathbf{P}$. As shown in [137, Theorem 6.27], for every irreducible and aperiodic transition matrix $\mathbf{P}$ there exists a stationary distribution $\underline{\boldsymbol{\lambda}}$. See also the work of Betancourt [17] for an extended discussion.

Ergodicity

If a Markov chain has an irreducible and aperiodic transition matrix $\mathbf{P}$ and thus a stationary distribution $\underline{\boldsymbol{\lambda}}$, the ergodicity theorem states that for a measurable function f

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{\lambda}_i) = \int_S f(\boldsymbol{\lambda})\,\mathrm{d}\mu(\underline{\boldsymbol{\lambda}}), \tag{3.24}$$

that is the average over an infinite run of the Markov chain in time i equals a weighted average over the state space S by the stationary distribution $\underline{\boldsymbol{\lambda}}$. The concept of ergodicity is deeply rooted in statistical physics, but was also extensively studied in mathematics, see e.g. [135] for a historical overview or the work of Peters and Gell-Mann [160] for its application to decision theory.

Sampling by construction of transition matrix

Taking the previous results together, we can sample from a desired distribution by constructing a Markov Chain with a certain transition matrix, such that the resulting stationary distribution resembles exactly the desired distribution, giving rise to the name *Markov chain Monte Carlo*.

3.4.3. Markov chain Monte Carlo algorithms

Distributional form

Although we are particularly interested in transition matrices for distributions that have the form of the posterior in a Bayesian analysis for a continuous[4] parameter

$$P_{\Theta\,|\,X}(\theta\,|\,x) = \frac{P_{X\,|\,\Theta}(x\,|\,\theta)\,P_\Theta(\theta)}{\int_\Omega P_{X\,|\,\Theta}(x\,|\,\theta)\,P_\Theta(\theta)\,\mathrm{d}\mu(\theta)}, \tag{3.25}$$

in this subsection we will see how to construct the transition matrix for the general case of a random variable Λ that has a distribution of the form

$$P_\Lambda(\lambda) = \frac{1}{Z}\mathrm{e}^{-\beta E(\lambda)}, \tag{3.26}$$

with $\beta > 0$ constant and Z given by the usually intractable integral

$$Z = \int_\Omega \mathrm{e}^{-\beta E(\lambda)}\,\mathrm{d}\mu(\lambda). \tag{3.27}$$

[4] Again, there is no loss of generality when assuming a continuous parameter. See footnote around

In physics, this distribution is called Boltzmann distribution and plays a major role in statistical mechanics. Consequently, the approaches presented in the following are heavily influenced by ideas from and applications to physics.

By setting

$$E(\theta) = -\ln\Big(P_{X\,|\,\Theta}(x\,|\,\theta)\,P_{\Theta}(\theta)\Big) \tag{3.28}$$

the results given below can be applied to the case of posterior inference as well.

Metropolis-Hastings

Metropolis et al. [131] gave the first publicly known description of a Markov chain Monte Carlo algorithm, also discussing the advantage over the simple Monte Carlo method. Their algorithm was later improved by Hastings [88], resulting in the so-called Metropolis-Hastings algorithm.

Derivation

The derivation works as follows: we impose the reversibility condition

$$P_{\Lambda}(\lambda_i)\,P_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j) = P_{\Lambda}(\lambda_j)\,P_{\Lambda_j|\Lambda_i}(\lambda_j|\lambda_i), \tag{3.29}$$

also known as *detailed balance*. As a result, the Markov chain has a stationary distribution [137, Lemma 6.24] and we can apply the ergodic theorem. Then we split the transition probabilities into two parts,

$$P_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j) = G_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j)\,A_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j), \tag{3.30}$$

the generation and acceptance probability. The proposed choice of Hastings [88] is

$$A_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j) = \min\left(1, \frac{P_{\Lambda}(\lambda_i)}{P_{\Lambda}(\lambda_j)}\,\frac{G_{\Lambda_j|\Lambda_i}(\lambda_j|\lambda_i)}{G_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j)}\right), \tag{3.31}$$

which is only based on ratios of $P_{\Lambda}(\lambda)$, such that the unknown quantity Z in (3.26) cancels out. Checking that (3.29) is fulfilled, is straightforward by inspecting all cases. The resulting algorithm is shown in Algorithm 3.1.

Performance and derived algorithms

The practical performance of the algorithm largely depends on the choice of $G_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j)$, most importantly a low rate of rejections $\lambda_{i+1} = \lambda_i$ is desired. Starting from the Metropolis-Hastings algorithm, many other algorithms have been derived, e.g. the family of Gibbs sampling algorithms, see Geman and Geman [76] for the founding work and [74] for further discussion. In the next subsection we will however focus on a family of algorithms that borrow concepts from physics for the construction of generators $G_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j)$.

equation (2.2) for the general case that involves evaluation of a measure over the whole space.

Algorithm 3.1 Metropolis-Hastings algorithm

Require: Number of samples n, initial value λ_0, generator $G_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j)$
  for sample i in $\{1, \ldots, n\}$ do
    Draw λ distributed as $G_{\Lambda|\Lambda_{i-1}}(\lambda|\lambda_{i-1})$
    Compute $A_{\Lambda|\Lambda_{i-1}}(\lambda|\lambda_{i-1}) = \min\left(1, \frac{P_\Lambda(\lambda)}{P_\Lambda(\lambda_{i-1})} \frac{G_{\Lambda_{i-1}|\Lambda}(\lambda_{i-1}|\lambda)}{G_{\Lambda|\Lambda_{i-1}}(\lambda|\lambda_{i-1})}\right)$
    Draw u uniformly distributed in $[0, 1]$
    if $u \leq A_{\Lambda|\Lambda_{i-1}}(\lambda|\lambda_{i-1})$ then
      Set $\lambda_i = \lambda$
    else
      Set $\lambda_i = \lambda_{i-1}$
    end if
  end for

Hamiltonian Monte Carlo

Hamiltonian Monte Carlo algorithms are named after the field of Hamiltonian mechanics, by which they were inspired. They promise better performance than most simple Metropolis-Hastings variants for continuous variables, especially for hierarchical models, see Betancourt and Girolami [16] for details. We will later cover a hierarchical model in section 3.4.3.

Hamiltonian Monte Carlo is often abrevaded as HMC, which is sometimes used for *Hybrid Monte Carlo* as defined by Duane et al. [58] as well. Both, however, describe the same overall concept.

Hamiltonian dynamics

The key idea is to use a special mapping for the generator $G_{\Lambda_i|\Lambda_j}(\lambda_i|\lambda_j)$ by computing λ_i as the endpoint of a trajectory starting at λ_j,

$$\lambda_i = \mathcal{H}(\lambda_j|p, T) \tag{3.32}$$

with random initial conditions. The exact procedure is as follows: we consider again the case of a vector $\boldsymbol{\lambda}$ in the state space Ω and introduce vectors $\boldsymbol{p}$ with the same dimensions, called *momentum*. In a similar manner as in classical mechanics, we define the *Hamiltonian*

$$H(\boldsymbol{\lambda}, \boldsymbol{p}) = U(\boldsymbol{\lambda}) + K(\boldsymbol{p}) \equiv U(\boldsymbol{\lambda}) + \frac{1}{2}\, \boldsymbol{p}\, \mathbf{M}^{-1}\, \boldsymbol{p}, \tag{3.33}$$

with U from (3.28) and the kinetic energy K. The dynamics of a system of particles in classical mechanics is described by the following Hamiltonian equations of motion, which we will adopt as well:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\lambda}(t) = \quad \frac{\partial}{\partial \boldsymbol{p}} H(\boldsymbol{\lambda}, \boldsymbol{p}) \tag{3.34}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{p}(t) = -\frac{\partial}{\partial \boldsymbol{\lambda}} H(\boldsymbol{\lambda}, \boldsymbol{p}) \tag{3.35}$$

Algorithm 3.2 Hamiltonian Monte Carlo algorithm

Require: Number of samples n, initial value $\boldsymbol{\lambda}_0$, number of integration steps L, integration steps size ϵ, energy functions U and K
  for sample i in $\{1, \ldots, n\}$ do
    Set $\boldsymbol{\lambda} = \boldsymbol{\lambda}_{i-1}$
    Set $\boldsymbol{p}_0 \sim \mathcal{N}(0, 1)$
    Set $\boldsymbol{p} = \boldsymbol{p}_0 - \frac{\epsilon}{2} \nabla U(\boldsymbol{\lambda})$
    for step j in $\{1, \ldots, L\}$ do
      Set $\boldsymbol{\lambda} = \boldsymbol{\lambda} + \epsilon \boldsymbol{p}$
      if $j \neq L$ then
        Set $\boldsymbol{p} = \boldsymbol{p} - \epsilon \nabla U(\boldsymbol{\lambda})$
      end if
    end for
    Set $\boldsymbol{p} = -\left(\boldsymbol{p} - \frac{\epsilon}{2} \nabla U(\boldsymbol{\lambda})\right)$
    Set $A_{\Lambda|\Lambda_{i-1}}(\lambda|\lambda_{i-1}) = \exp\left(U(\boldsymbol{\lambda}_{i-1}) - U(\boldsymbol{\lambda}) + K(\boldsymbol{p}_0) - K(\boldsymbol{p})\right)$
    Draw u uniformly distributed in $[0, 1]$
    if $u \leq A_{\Lambda|\Lambda_{i-1}}(\lambda|\lambda_{i-1})$ then
      Set $\lambda_i = \lambda$
    else
      Set $\lambda_i = \lambda_{i-1}$
    end if
  end for

Finally, we can specify (3.32) precisely as

$$\mathcal{H}(\lambda_j | \, p, T) = \boldsymbol{\lambda}(T) \tag{3.36}$$

as the solution to (3.34) and (3.35), provided the initial conditions

$$\lambda(0) = \lambda_j \qquad \boldsymbol{p}(0) \sim \mathcal{P}, \tag{3.37}$$

that is a random value for the momentum according to some distribution $\mathcal{P}$. The solution to the differential equations are usually provided by numerical integration, see Neal [140] for details and an overview. Algorithm 3.2 shows a specific version with a *leapfrog* integration scheme and a standard Gaussian distribution $\mathcal{P}$ as given by Neal [140].

A main property of Hamiltonian mechanics is that the Hamiltonian (3.33) is a conserved quantity for all tuples $(\boldsymbol{\lambda}, \boldsymbol{p})$ along the trajectory. Thus, if one was to use a ratio of probabilities $P_{\Lambda,P}(\boldsymbol{\lambda}, \boldsymbol{p}) = \frac{1}{Z} \exp(-\beta \, H(\boldsymbol{\lambda}, \boldsymbol{p}))$ for sampling (Λ, P) by a Metropolis-Hastings algorithm, that ratio would equal one for a perfect simulation of the trajectory, resulting in zero rejection rate. As, however, numerical simulation of the trajectory is imperfect and the marginal distribution of Λ is the one of interest,

Rejection rate

the practical rejection rate is above zero. An example for the latter is the mapping $(\boldsymbol{\lambda}, \boldsymbol{p}) \mapsto (\boldsymbol{\lambda}, -\boldsymbol{p})$ that leaves the Hamiltonian (3.33) invariant, but produces no new value of $\boldsymbol{\lambda}$. See again Neal [140] for an extended discussion.

Geometrical interpretation

As an alternative to the connection to classical mechanics, there are also treatments of the subject from a differential geometric point of view, see Betancourt et al. [19] and Barp et al. [8].

Intuitively speaking, trajectories of Hamiltonian Monte Carlo samplers are more self-avoiding in contrast to simple Metropolis-Hastings variants that slowly diffuse like a random walk.

Parameter choice

The right choice of the hyperparameters is essential for practical performance, see e.g. Betancourt, Byrne, and Girolami [15] for recommendations on step size, while Neal [140] discusses the choice of the mass matrix $\mathbf{M}^{-1}$, recommending to approximate the covariance matrix of $p_\Lambda(\lambda)$.

The no-U-turn sampler

Homan and Gelman [93] developed an HMC algorithm called *no-U-turn sampler* (NUTS) that automatically controls the number of steps and the step size of the numerical integration, thus freeing the user from tuning those parameters.

Controlling the number of steps

The main idea for controlling the number of steps is to check whether an additional step would bring the algorithm back to the origin of the path and stopping the integration, if that is the case. This way of avoiding a U-turn of the sampler is reflected in the name of the algorithm accordingly.

Optimizing the steps size

The step size is optimized during the warm-up phase of the sampling using an initial guess that gets adapted to match a desired acceptance rate of the proposals. For the production run of the sampler the value found in the warm-up is used.

Algorithm 3.3 shows the version given by Homan and Gelman [93] in their work, which also contains more information on the technical details. From the listing, we can see that the sophistication has largely grown compared to the simple Metropolis-Hastings algorithm shown in Algorithm 3.1. However, the end user strongly benefits from the additional effort, as usually no manual tuning of the inference is necessary, greatly reducing the amount of manual effort and expertise needed.

Potential issues and diagnosis

Convergence

When applied in practice, Markov-Chain Monte Carlo methods have potential issues as well. The first and foremost is that we can draw only a finite number of samples and thus we have to check whether the sequence has already converged. This is typically achieved by starting multiple chains with different starting points and then checking whether they are mixing well and have reached stationarity. Gelman and Rubin [73] developed a statistic to diagnose this behavior, see Gelman et al. [74] for further discussion.

Another potential problem is that of correlated samples, which reduces the effective sample size. In order to avoid strong dependence on the initial values, the first part of a sampled chain, called warm-up, is often discarded and sometimes the chain is even thinned afterwards, see again Gelman et al. [74] for further discussion. Correlation

For Hamiltonian Monte Carlo (HMC) methods in particular, there are additional diagnosis methods, namely checking the energy distributions of transitions and whether the numerical integration diverged, see the work of Betancourt [18] for more details. HMC diagnosis

Example case: Hierarchical beta-binomial model

To illustrate the concepts discussed above, we will extend the example from section 3.1 to the case of N distinct sets of observations, consisting of n_i trials and x_i 'successes'. Thus, we need to understand Θ as $\Theta = \{\Theta_i\}_{i=1}^{N}$.

We start with a modified likelihood by again assuming identical probability for success and independence of the trials, but for each set of observations a separate binomial distribution Model formulation

$$P_{X\,|\,\Theta}(x\,|\,\theta) = \prod_{i=1}^{N} \binom{n_i}{x_i} \theta_i^{x_i} \, (1-\theta_i)^{n-x_i}. \tag{3.38}$$

Next, we keep the same prior, a beta distribution, for each component, such that the prior density function is

$$\pi_{\Theta}(\theta) = \prod_{i=1}^{N} \frac{\theta_i^{\Phi-1} \, (1-\theta_i)^{\Psi-1}}{B(a,b)}, \tag{3.39}$$

but with shared nuisance parameters Φ and Ψ, which are random variables as well.

This model choice has two consequences. First, the nuisance parameters Φ and Ψ require priors, so-called hyperpriors, which are chosen as half-normal distributions with density functions

$$\pi_{\Phi}(\phi) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}} \exp\left(-\frac{\phi^2}{2\sigma_{\Phi}^2}\right), \tag{3.40}$$

$$\pi_{\Psi}(\psi) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}} \exp\left(-\frac{\psi^2}{2\sigma_{\Psi}^2}\right), \tag{3.41}$$

where $\phi, \psi, \sigma_{\Phi}, \sigma_{\Psi} \geq 0$.

Second, the resulting model is a hierarchical model, which allows for information exchange between parameters for different sets of observations via the common hyperparameters. Hierarchical models have benefits both from a theoretical and practical point of view: the unified treatment of several groups of observations is in accordance to the intuition that knowledge about one set is useful for the others,

while the smaller number of parameters compared to modelling each set separately reduces the danger of overfitting and allows for meaningful inference even if some or all data sets consist of only few observations. See [74, chapter 5] for a more detailed discussion of hierarchical models in general and the particular example.

Example data

Even though we could study the model in its general form, we consider here a concrete example case where we assume $N = 3$ with $x = (7, 23, 14)$ and $n = (10, 100, 50)$. Additionally, we fix $\sigma_\alpha = \sigma_\beta = 7$ for an approximately flat prior.

Inference by NUTS

Fixing all values numerically allows us to do Bayesian inference by the NUTS algorithm of Homan and Gelman [93]. In our experiments, we use the implementation of Salvatier, Wiecki, and Fonnesbeck [176].

Results

Figure 3.2 shows the results. The sample from the prior in Figure 3.2a approaches the uniform distribution from Figure 3.1a. The sampling trace after removing the warm-up is shown in Figure 3.2b for illustrational purposes, diagnostics should not be done by visual inspection, except for obvious cases of problems. The posterior samples are shown in Figure 3.2c with the maximum likelihood estimates (3.13) for each set of observations shown in black line color for reference.

Partial pooling

We can clearly see a useful property of hierarchical models here that is known as partial pooling. Due to the common hyperparameters information from other sets of observations is taken into account as well. In our case, this e.g. means that the estimate for the first set of observations is biased away from the maximum likelihood estimates, as the other data sets carry contradicting information. Moreover, the different sizes of sets of observations are taken into account as well, yielding different strengths of the effect for the different parameter estimates. See again [74, chapter 5] for a more detailed discussion.

Comparison of sets of observations

Often, hierarchical models like the one shown here are used to compare slightly different sets of observations for which a comparison on a data level is not meaningful, see e.g. Calandra et al. [32] and Pedergnana et al. [157] for applications in our own work.

3.4.4. Sequential Monte Carlo

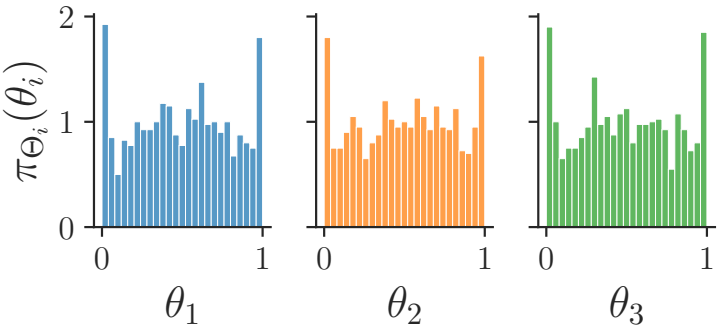
In contrast to the Markov Chain Monte Carlo (MCMC) methods discussed above, the Sequential Monte Carlo (SMC) methods aim to infer a sequence of distributions [116]. Usually the distributions are encoded as weighted samples, sometimes called *particles*. Del Moral, Doucet, and Jasra [51] give an overview on different sampling techniques.

Particle filters

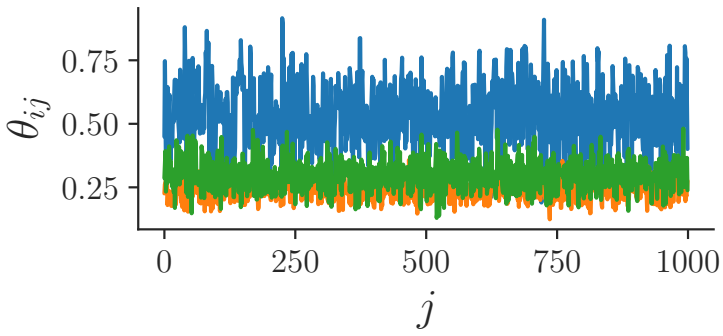
When SMC algorithms are applied to the *filtering problem*, i.e., estimating the hidden state of a system from noisy observations in an online setting, these methods are often called particle filters. See e.g. [34] as textbook for an introduction. Due to a similar mathematical structure of the problems, there is a link to the work of Feynman and Kac on path integrals and stochastic processes [50].

Combination with HMC

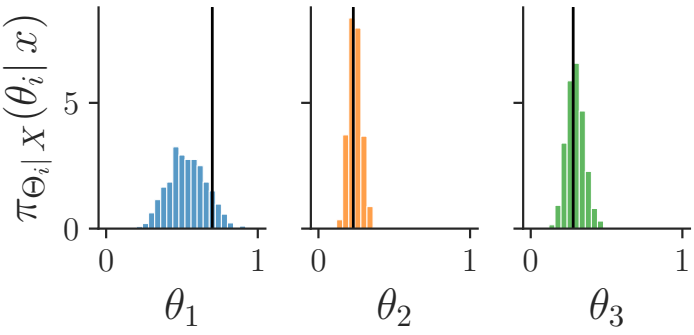
Buchholz, Chopin, and Jacob [31] proposed to use the transition kernels of



(a) Prior density.



(b) Sampling trace after warm-up.



(c) Posterior density and maximum likelihood estimates (3.13) shown in black line color. Notice the effect of partial pooling in the leftmost panel.

Figure 3.2.: Prior, sampling trace and posterior for a hierarchical beta-binomial model on example data $N = 3$, $x = (7, 23, 14)$ and $n = (10, 100, 50)$. See text in section 3.4.3 for more details.

HMC in SMC, thus combining the benefits of both approaches.

3.5. Recursive Inference or online updating

When the data X are not completely available, but continue to arrive steadily one after another, the inference takes places in a so-called *online* setting. Particle based methods are ideally suited for that task by construction. With MCMC methods one usually reruns the inference for each new dataset. As an alternative, Hooten, Johnson, and Brost [94] propose an algorithm for updating samples with MCMC methods.

3.6. Parallelization

If the computational load of a task is high, the question whether we can gain some speed-up by parallalelization arises naturally.

Multiple chains

For MCMC methods one usually attempts to compute several chains independently anyhow, to use them as a diagnostics of convergences. Thus the problem is trivially parallelizable at this level.

Finite Decomposability

If the computation of a single chain is too costly, however, there are problems with the decomposability. Although a single atomic operation for an MCMC algorithm consists in computing a proposal and rejecting or accepting it, which would allow for a large scale parallelization down to a fine level, the serial correlation between samples would invalidate the results when splitting the chain into too small parts. Thus, there is an upper limit for the parallelization of chains by splitting them into smaller ones.

Implementations

Holbrook et al. [92] present a parallel version of HMC and Zhou [213] a parallel version of SMC. For parallelization of particle filters see [46] and [117].

Algorithm 3.3 No-U-Turn sampler with dual averaging

Require: Initial value $\boldsymbol{\lambda}_0$, δ, $U, K, M, M^{\text{adapt}}$
  Set $\epsilon_0 = \text{FindReasonableEpsilon}(\Lambda), \mu = \log(10\epsilon_0), \bar{\epsilon}_0 = 1, \bar{H}_0 = 0, \gamma = 0.05, t_0 = 10, \kappa = 0.75$.
  for $m = 1$ to M do
    Sample $\boldsymbol{p}_0 \sim \mathcal{N}(0, I)$.
    Resample $u \sim \mathcal{U}([0, \exp\{U(\boldsymbol{\lambda}_{m-1}) - K(\boldsymbol{p}_0)\}])$
    Initialize $\boldsymbol{\lambda}_- = \boldsymbol{\lambda}_{m-1}$, $\boldsymbol{\lambda}_+ = \boldsymbol{\lambda}_{m-1}$, $\boldsymbol{p}_- = \boldsymbol{p}_0, \boldsymbol{p}_+ = \boldsymbol{p}_0, j = 0, \boldsymbol{\lambda}_m = \boldsymbol{\lambda}_{m-1}, n = 1, s = 1$.
    while $s = 1$ do
      Choose a direction $v_j \sim \mathcal{U}(\{-1, 1\})$.
      if $v_j = -1$ then
        $\boldsymbol{\lambda}_-, \boldsymbol{p}_-, -, -, \boldsymbol{\lambda}', n', s', \alpha, n_\alpha \leftarrow \text{BuildTree}(\boldsymbol{\lambda}_-, \boldsymbol{p}_-, u, v_j, j, \epsilon_{m-1}, \boldsymbol{\lambda}_{m-1}, \boldsymbol{p}_0)$.
      else
        $-, -, \boldsymbol{\lambda}_+, \boldsymbol{p}_+, \boldsymbol{\lambda}', n', s', \alpha, n_\alpha \leftarrow \text{BuildTree}(\boldsymbol{\lambda}_+, \boldsymbol{p}_+, u, v_j, j, \epsilon_{m-1}, \boldsymbol{\lambda}_{m-1}, \boldsymbol{p}_0)$.
      end if
      if $s' = 1$ then
        With probability $\min\{1, \frac{n'}{n}\}$, set $\boldsymbol{\lambda}_m \leftarrow \boldsymbol{\lambda}'$.
      end if
      $n \leftarrow n + n'$.
      $s \leftarrow s' \mathbb{I}[(\boldsymbol{\lambda}_+ - \boldsymbol{\lambda}_-) \cdot \boldsymbol{p}_- \geq 0] \mathbb{I}[(\boldsymbol{\lambda}_+ - \boldsymbol{\lambda}_-) \cdot \boldsymbol{p}_+ \geq 0]$.
      $j \leftarrow j + 1$.
    end while
    if $m \leq M^{\text{adapt}}$ then
      Set $\bar{H}_m = \left(1 - \frac{1}{m + t_0}\right) \bar{H}_{m-1} + \frac{1}{m + t_0}(\delta - \frac{\alpha}{n_\alpha})$.
      Set $\log \epsilon_m = \mu - \frac{\sqrt{m}}{\gamma} \bar{H}_m, \log \bar{\epsilon}_m = m^{-\kappa} \log \epsilon_m + (1 - m^{-\kappa}) \log \bar{\epsilon}_{m-1}$.
    else
      Set $\epsilon_m = \bar{\epsilon}_{M^{\text{adapt}}}$.
    end if
  end for

  function BuildTree($\boldsymbol{\lambda}, \boldsymbol{p}, u, v, j, \epsilon, \boldsymbol{\lambda}_0, \boldsymbol{\lambda}_0$)
  if $j = 0$ then
    $\boldsymbol{\lambda}', \boldsymbol{p}' \leftarrow \text{Leapfrog}(\boldsymbol{\lambda}, \boldsymbol{p}, v\epsilon)$.
    $n' \leftarrow \mathbb{I}[u \leq \exp\{U(\boldsymbol{\lambda}') - K(\boldsymbol{p}')\}]$.
    $s' \leftarrow \mathbb{I}[u < \exp\{\Delta_{\max} + U(\boldsymbol{\lambda}') - K(\boldsymbol{p}')\}]$.
    return $\boldsymbol{\lambda}', \boldsymbol{p}', \boldsymbol{\lambda}', \boldsymbol{p}', \boldsymbol{\lambda}', n', s', \min\{1, \exp\{U(\boldsymbol{\lambda}') - K(\boldsymbol{p}') - U(\boldsymbol{\lambda}_0) - K(\boldsymbol{p}_0)\}\}, 1$.
  else
    $\boldsymbol{\lambda}_-, \boldsymbol{p}_-, \boldsymbol{\lambda}_+, \boldsymbol{p}_+, \boldsymbol{\lambda}', n', s', \alpha', n'_\alpha \leftarrow \text{BuildTree}(\boldsymbol{\lambda}, \boldsymbol{p}, u, v, j - 1, \epsilon, \boldsymbol{\lambda}_0, \boldsymbol{p}_0)$.
    if $s' = 1$ then
      if $v = -1$ then
        $\boldsymbol{\lambda}_-, \boldsymbol{p}_-, -, -, \boldsymbol{\lambda}'', n'', s'', \alpha'', n''_\alpha \leftarrow \text{BuildTree}(\boldsymbol{\lambda}_-, \boldsymbol{p}_-, u, v, j - 1, \epsilon, \boldsymbol{\lambda}_0, \boldsymbol{p}_0)$.
      else
        $-, -, \boldsymbol{\lambda}_+, \boldsymbol{p}_+, \boldsymbol{\lambda}'', n'', s'', \alpha'', n''_\alpha \leftarrow \text{BuildTree}(\boldsymbol{\lambda}_+, \boldsymbol{p}_+, u, v, j - 1, \epsilon, \boldsymbol{\lambda}_0, \boldsymbol{p}_0)$.
      end if
      With probability $\frac{n''}{n' + n''}$, set $\boldsymbol{\lambda}' \leftarrow \theta''$.
      Set $\alpha' \leftarrow \alpha' + \alpha''$, $n'_\alpha \leftarrow n'_\alpha + n''_\alpha$.
      $s' \leftarrow s'' \mathbb{I}[(\boldsymbol{\lambda}_+ - \boldsymbol{\lambda}_-) \cdot \boldsymbol{p}_- \geq 0] \mathbb{I}[(\boldsymbol{\lambda}_+ - \boldsymbol{\lambda}_-) \cdot \boldsymbol{p}_+ \geq 0]$
      $n' \leftarrow n' + n''$
    end if
    return $\boldsymbol{\lambda}_-, \boldsymbol{p}_-, \boldsymbol{\lambda}_+, \boldsymbol{p}_+, \boldsymbol{\lambda}', n', s', \alpha', n'_\alpha$.
  end if

4. Parallelization of the no-U-turn sampler

While in chapter 3 generic inference algorithms were covered, here we will turn our attention to more specific challenges and present a solution for them.

4.1. Inference runtime on large data sets as a practical challenge

In some applications, there are data sets with many observations of equal importance, which is generally considered beneficial, as there is potentially a lot of information contained in the observations. However, the resulting runtime of preferred inference algorithms like NUTS can be high, which is a challenge especially in online settings, see chapter 6 and chapter 9. Thus, the question is how to reduce the runtime of the inference on large data sets without sacrificing ideally any or at least only mild amounts of inference quality.

Other approaches

Of course, for all cases it is beneficial to have well designed and tuned implementations of the algorithms used, see chapter 5 for further discussion on general issues of programming Bayesian inference. For the more specific case of large data sets, there are, amongst others, the possibilities to speed up the programs on GPU architectures [92] or through subsampling of data, see [170] for further discussion.

4.2. The paraNUTS algorithm

A typical approach to speeding up algorithms is to use parallel computing, where one splits the task into (ideally) independent subtasks, –which can be solved in parallel– and later on combines the results of those to the final result. Following this principle, we propose to run the inference on subsets by MCMC algorithms in parallel and to combine the resulting posterior samples accordingly.

Related work

A similar approach was also taken by several others: while Scott et al. [181] suggest a weighted average combine the resulting posterior samples, Wang and Dunson [205] and Neiswanger, Wang, and Xing [142] use parametric and non-parametric representations which express the full posterior as a product, Minsker et al. [132] compute an embedding in order to denoise the samples and Nemeth and Sherlock [144] approximate the posterior log-density by a Gaussian Process.

Idea

We propose a novel approach of this kind. For the inference on the subsets, there are no alterations to the usual case in our approach, such that we can use the NUTS algorithm straightforwardly, see section 3.4.3. Running the inference on the subsets results in different sets of parameter samples for each subset. When combining the samples, we use the computationally cheap Metropolis criterion, see section 3.4.3, but using all data for computing the likelihood, thus ensuring that in the limit of many merged samples we sample from the full model evaluated on all data. In the following we will call this approach the *paraNUTS* algorithm.

Mathematical formulation

To make the formulation more precise we will stick to the following notation: we consider a model $\mathcal{M} = \mathcal{M}(\Theta)$ described by parameters Θ for observations X. Now we consider a partition $P = \{X_i\}$ of the observations such that

$$\emptyset \notin P \tag{4.1}$$

$$\bigcup_{X_i \in P} X_i = X \tag{4.2}$$

$$X_i \cap X_j = \emptyset \quad \forall\, X_i, X_j \neq X_i \in P, \tag{4.3}$$

where $\emptyset$ denotes the empty set and we have taken the definition of a partition from [118]. During the parallel steps we sample

$$\hat{\theta}_i \sim P_{\hat{\Theta}_i \,|\, X_i}(\hat{\theta}_i \,|\, x_i) \tag{4.4}$$

on each of the X_i and finally we sample

$$\theta \sim P_{\Theta \,|\, X}(\theta \,|\, x) \tag{4.5}$$

on the full data X with $\hat{\theta}_i$ as candidates for the Metropolis criterion.

Inspiration

The approach was inspired from the method of parallel tempering [191] [96] [64] [61] or replica exchange molecular dynamics [190] used in computational physics, where several instances of a system with slightly different setups (e.g. temperature) are simulated (potentially in parallel) and particles are swapped between systems to improve the simulations. See the work by Neal [141] for an application in statistical computing.

Using the map-reduce paradigm

When an algorithm can be formulated in terms of so-called *map* and *reduce* operations, it is usually simple to leverage parallelization in an implementation. Here, *map* refers to applying a unary operation $f(\cdot)$ to all elements in a collection or data structure, which can be done in an embarrassingly parallel fashion. *Reduce* refers to applying a certain class of binary operations $b(\cdot, \cdot)$ repeatedly on elements in a collection or data structure, until only a single element is returned. More precisely, the operation $b(\cdot, \cdot)$ and the set $\mathcal{S}$ of elements on which it should act is commonly required to be a monoid [133], i.e. the map $b : \mathcal{S} \times \mathcal{S} \to \mathcal{S},\ (s_1, s_2) \mapsto s_1 \circ s_2$ has to fulfill

$$(s_1 \circ (s_2 \circ s_3)) = ((s_1 \circ s_2) \circ s_3)\ \forall\, s_1, s_2, s_3 \in \mathcal{S} \tag{4.6}$$

$$\exists\, e \in \mathcal{S} : e \circ s = s \circ e\ \forall\, s \in \mathcal{S}. \tag{4.7}$$

Algorithm 4.1 paraNUTS

Require: Number of samples n, initial value λ_0, NUTS
  Partition data into $P = \{X_i\}_{i=1}^{p}$
  for sample i in $\{1, \ldots, p\}$ do
    Compute n samples $\hat{\theta}_i$ on X_i by NUTS starting at λ_0 ▷ Map
  end for
  repeat ▷ Reduce
    for i in $\{1, \ldots, n\}$ do
      Consider $\lambda_{i,j}$ from $\hat{\theta}_j = \{\lambda_{lj}\}_{l=1}^{n}$ and $\lambda_{i,k}$ from $\hat{\theta}_k = \{\lambda_{mk}\}_{m=1}^{n}$ with $k \neq j$
      Compute $A_{\Lambda_{i,j}|\Lambda_{i,k}}(\lambda_{i,j}|\lambda_{i,k}) = \min\left(1, \frac{P_\Lambda(\lambda)}{P_\Lambda(\lambda_{i-1})}\right)$
      Draw u uniformly distributed in $[0, 1]$
      if $u \leq A_{\Lambda_{i,j}|\Lambda_{i,k}}(\lambda_{i,j}|\lambda_{i,k})$ then
        Set $\lambda = \lambda_{i,j}$
      else
        Set $\lambda = \lambda_{i,k}$
      end if
      Store λ in $\hat{\theta} = \{\lambda_i\}_{i=1}^{n}$
    end for
  until all $\hat{\theta}$ are reduced into one

If these conditions are fulfilled, $b(\cdot, \cdot)$ can be applied in parallel to pairs of elements in a collection or data structure, until a single value $s \in \mathcal{S}$ is left. Applications that are designed after the Map-Reduce paradigm can be scaled to a huge number of computations in parallel, see e.g [112] or [48] for further discussion or [211] for a framework to implement those applications.

In terms of the map-reduce paradigm, the sampling from the posterior on subsets in (4.4) can be considered as the map operation and the sampling from the posterior on the full data set in (4.5) as the reduce. However, as the ratios and the stochastic nature of the Metropolis criterion, see Algorithm 3.1, break the associativity in (4.7), the resulting structure is a unital magma only [198] [120]. Nevertheless, the operation still allows for parallel execution and yields the desired result in the sense of stochastic convergence.

Description, implementation and availability

Algorithm 4.1 shows a listing of the paraNUTS algorithm in pseudo-code.

We used the Julia programming language [20] for an implementation that is built upon the Turing.jl [71] package for Bayesian inference and the folds [6] package for the parallel execution of the map and reduce steps.

Source code and data is available at https://github.com/KonstantinBob/paraNUTS and a corresponding publication is in preparation.

4.3. Use case study

As an examplary use case, we studied a Gaussian process (GP) model, see the work of Rasmussen and Williams [171] for a detailed coverage of the topic. See also subsection 6.4.1 for a possible application of a Gaussian process. The process models pairs of data points $\{(x_i, y_i)\}_{i=1}^n$. A key component is that the model requires a notion of distance between two realizations of the first component, x_i and x_j. Thus, for applications, the right choice of a distance function $d(\cdot, \cdot)$ is important [171].

Mathematical formulation

We assume that a distance matrix D with entries $d_{ij} = d(x_i, x_j)$ is precomputed. The mathematical formulation of the model used is given as follows

$$\sigma^2 \sim \mathcal{LN}(0, 1) \tag{4.8}$$
$$\phi \sim \mathcal{IG}(3, 0.5) \tag{4.9}$$
$$\mu \sim \mathcal{N}(0, 1) \tag{4.10}$$
$$\Sigma = \exp\left(-\frac{D^2}{\phi^2}\right) + \sigma^2 I \tag{4.11}$$
$$y \sim \mathcal{N}(\mu, \Sigma), \tag{4.12}$$

where $\mathcal{LN}(\mu, \sigma)$ denotes the lognormal distribution with location parameter μ and scale parameter σ, $\mathcal{IG}(\alpha, \beta)$ denotes the inverse-gamma distribution with shape parameter α and scale parameter β. $\mathcal{N}(\mu, \sigma)$ denotes the Gaussian distribution with mean μ and standard deviation σ and I the identity matrix.

Data used

As data $\{(x_i, y_i)\}_{i=1}^n$ we used samples from the prior predictive distributions to avoid problems with modeling error. A data set with $n = 100$ is shown in Figure 4.1 for illustration.

Setup

The study was performed on Debian 10 machine with a AMD Ryzen 5 3600 6-Core Processor @ 2196MHz, allowing for parallel execution of 12 threads, and 32 GiB (2 × 16 GiB) of DIMM DDR4 Synchronous @ 2400 MHz main memory. We used Julia in version 1.6.2 and Turing.jl in version 0.17.4.

Results

The kernel-density estimates of the posterior distributions inferred by NUTS and paraNUTS on $k = 2$ and on $k = 4$ partitions are shown in Figure 4.2. Concerning the quality of the results, one finds that for all parameters the samples from both runs of paraNUTS have very similar central tendencies as those from NUTS, indicating good agreement. For the parameter ϕ (Figure 4.2a) they have even a similar shape, indicating similar results from both methods, independently of the number of partitions k. A widening of the distributions can be observed for the parameter σ^2 (Figure 4.2b), which is even more pronounced for the parameter μ (Figure 4.2c). In both cases the widening gets stronger with a higher number of partitions k.

Thus, one can conclude that estimates from paraNUTS on different partitions can have a greater uncertainty than those from a run with NUTS on the full data set. This, however, can be explained by the fact that the estimates for some parameters

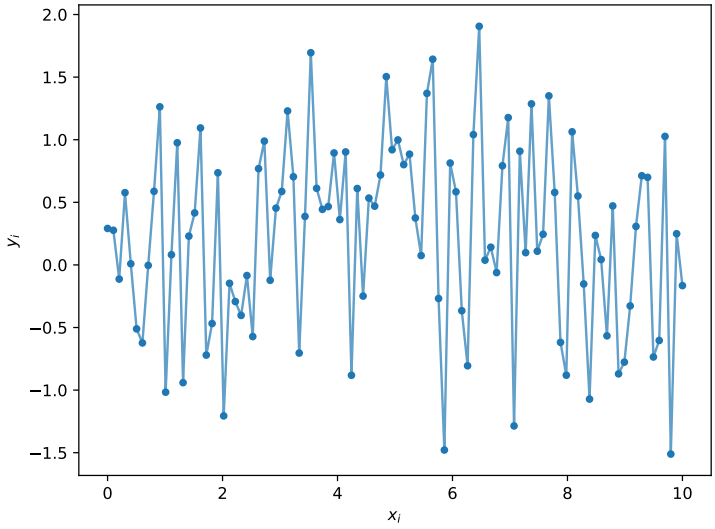
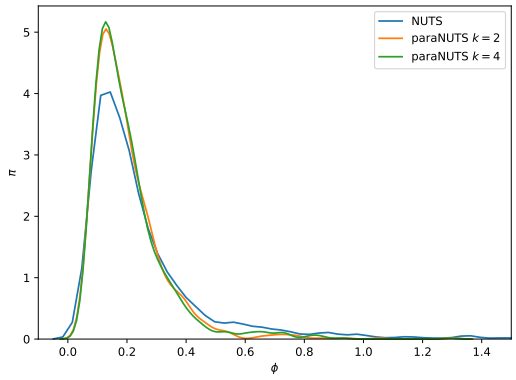


Figure 4.1.: Data used for inference by a Gaussian process model with $n = 100$ observations (blue dots). The straight lines are just used for guiding the eye.

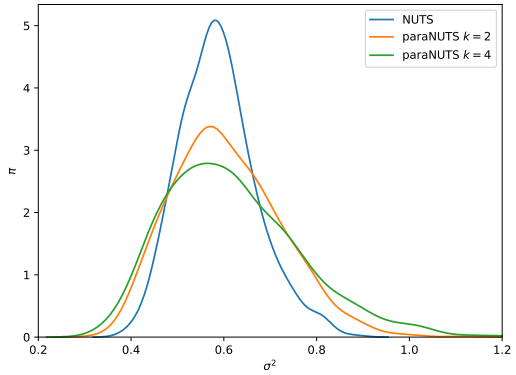
depend more heavily on the partitioning. E.g. in this concrete case, the parameter ϕ that measures a typical length scale for distances between the x_i, is less influenced by the partitioning than the parameter μ that measures the location of the y_i.

As a side note, it is worth to mention that this propagation of uncertainty is one of the beneficial characteristics of the Bayesian approach, as shown in section 2.4.

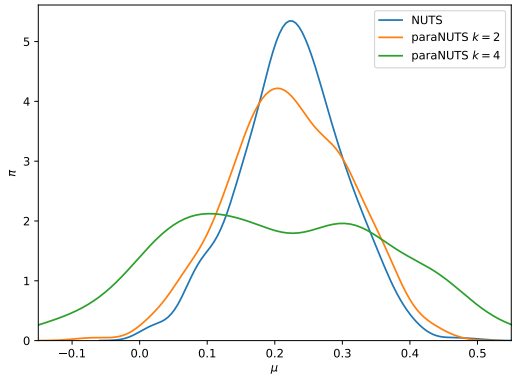
Figure 4.3 shows the scalability of inference by paraNUTS for two data sets with sizes $n = 140$ and $n = 250$. In both cases, up to a certain number of threads used, the scaling works well. In the best case for $n = 140$ observations a speed-up $s \approx 6$ and for $n = 250$ a speed-up $s \approx 12$ could be achieved. For larger numbers of threads, the parallelization overhead becomes noticeable and starts to increase runtime again slightly. Note that due to evaluation of dense matrices with shape $n \times n$ in equation (4.11) the computational effort grows considerably between the examples, which can also be seen when comparing the measured runtimes between Figure 4.3a and Figure 4.3b. As this higher computational burden did not lead to a greater number of threads yielding minimum runtime, one can conclude that model evaluation is not the limiting factor here. Thus, it is rather the case that the other tasks like creation of models and chains, model tuning and the reduce step start to dominate for a higher number of threads used. Those, however, do not benefit strongly from parallelization, such that they lead to the above mentioned overhead that leads to increasing runtimes.



(a) Parameter ϕ

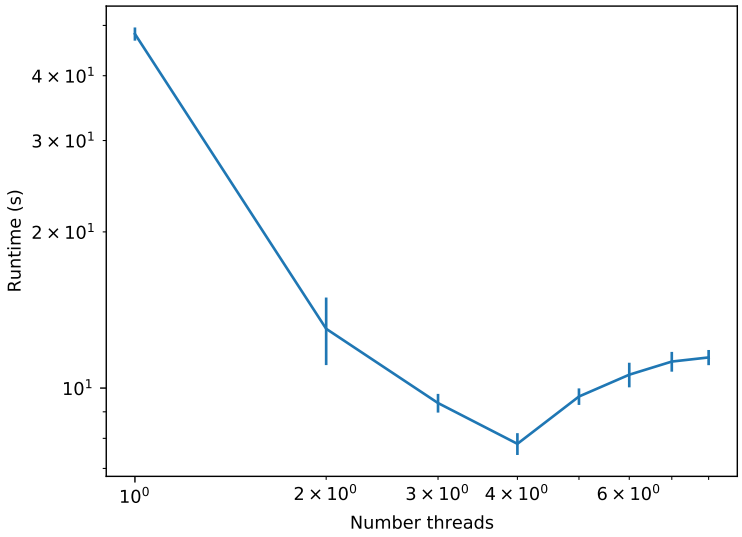


(b) Parameter σ^2

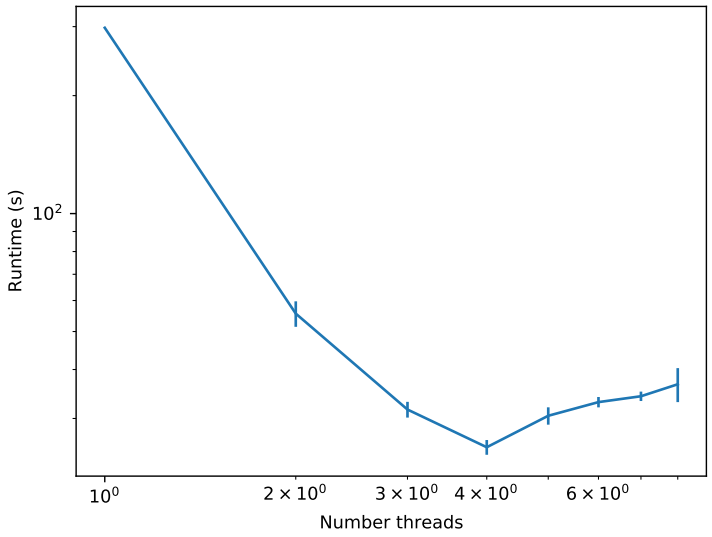


(c) Parameter μ

Figure 4.2.: Results from NUTS and paraNUTS. Kernel-density estimates of samples from standard NUTS (blue color), paraNUTS (orange color) on $k = 2$ partitions, and paraNUTS (green color) on $k = 4$ partitions for the three model parameters ϕ, σ^2, and μ with $n = 100$. See text in section 4.3 for further details.



(a) Inference on a data set with $n = 140$ observations



(b) Inference on a data set with $n = 250$ observations

Figure 4.3.: Scalability of inference on a Gaussian process model by paraNUTS. The graph show the runtime in seconds as a function of the number of threads used for the parallel execution, both on a logarithmic scale. Several runs were averaged, the curve shows mean and standard deviation. For both $n = 140$ and $n = 250$, the linear decrease at the left indicates that the scaling work wells until the optimal number of threads is hit. Afterwards, the runtime increases again slightly, indicating that the parallelization overhead is now larger than the gain in speed. See text in section 4.3 for further discussion.

5. Software libraries for probabilistic programming

In this work, we have so far dealt mainly with abstract concepts, such as mathematical formulations and algorithms. This chapter now covers software that turns data and algorithms into concrete results.

5.1. Probabilistic programming

For practical application of the algorithms shown in chapter 3, it is recommended to use a software library which was tested for reliability and optimized for performance. Furthermore, those software libraries typically provide utility functions for diagnostics and plotting, and their usage helps in ensuring reproducibility of the results.

Automated inference

When using these packages, the user is often only required to specify the prior and likelihood, while posterior inference and predictive distributions are handled by the software package, coining the term *probabilistic programming*. The development of software packages that allow the user to perform his tasks on a high level of abstraction without being forced to take care of low level details, aims to have a situation similar to that in the field of deep learning, where certain frameworks have lowered the entry threshold for non-experts considerably.

Table 5.1 shows a selection of some software packages for probabilistic programming, which we will refer to in the following discussion of a probabilistic programming library's components and their design.

5.2. Library components and their design

In general, a software package for probabilistic programming contains components for the model specification, the inference and prediction algorithms, and data structures for results. We will shortly review some design choices among the packages shown in Table 5.1.

Model specification

For the model specification, there are two types of design choices: either the model is specified in a domain specific language (DSL) or in a general purpose programming language. The popular package Stan [37], e.g. uses is own DSL, whereas the PyMC3 package [176] and the Turing.jl package [71] use general purpose programming languages, i.e. Python [201] [200] and Julia [20] respectively.

Table 5.1.: Selection of some software packages for probabilistic programming

Package name	API for programming languages	Reference
Stan	Python, Julia, R, MATLAB, Mathematica, command line	[37]
PyMC3	Python	[176]
TensorFlow Probability	Python	[53]
Turing.jl	Julia	[71]
Gen	Julia	[47]
Rainier	Scala	[30]

Type system

The choice of the model specification language influences the type system used as well. The Stan package [37] uses a static type system in its DSL, while the PyMC3 package [176] and the Turing.jl package [71] rely on the type system of the general purpose programming languages they use accordingly. For a detailed review on type system and their pros and cons see e.g. the work of Cardelli [35].

Automatic differentiation

An important aid for the use of Hamiltonian Monte Carlo algorithms (see section 3.4.3 for details) is the availability of automatic differentiation, that is the automated calculation of derivatives of mathematical expressions in the source code. Gebremedhin and Walther [72] give a detailed review on the topic and point out the design requirements for software systems, which in turn influences the design of probabilistic programming packages as well. While the Stan package provides its own implementation of automatic differentiation, Turing.jl can use a variety of Julia packages for that purpose and both TensorFlow Probability and PyMC3 delegate the task to the frameworks by which their models are compiled, which are Tensorflow [1] and Theano [196] respectively.

Compilation of model

As already mentioned, software packages for probabilistic programming usually use compiled code for greater computational speed. Again, different design choices have been made: the Turing.jl package works in this respect like other Julia language packages and is compiled according to the procedures of the Julia language. The Stan package translates code to the C++ programming language and uses the respective compilation methods to generate machine code. As noted before, PyMC3 and TensorFlow Probability use other frameworks, which can produce code for both CPUs and GPUs.

Available algorithms

Of course, a package should provide many different algorithms for inference. However, many more advanced algorithms have hyperparameters, which are non-trivial to set correctly or at least sufficiently well for the desired outcome. Thus, many packages try to provide automatically tuned parameters, see e.g. the discussion

in the article on PyMC3 [176]. In particular, the Turing.jl package was designed in a modular fashion, such that new algorithms can easily be added as engines [71]. The Gen package [47] aims for similar functionality.

Data structures

Besides the model representation that may requires to support more complex operations like automatic differentiation, the input data and resulting parameter samples are usually modeled as multi-dimensional arrays representing the mathematical tensors and their realizations as e.g. scalars, vectors, and matrices. A convenient feature for the user is the availability of broadcasting, that is the ability of the program to infer meaningful ways to simulataneosly perform operations like addition on arrays with different shapes. The Python package numpy [85] and Tensorflow [1], e.g., provide this functionality. For more details on broadcasting see e.g. [85].

5.3. Architectures for parallel processing

As Bayesian inference problems can require large amounts of computation, there is of course an interest in parallel processing methods to speed up computations and to be able to make efficient use of modern hardware.

Multithreaded CPU

As discussed in section 3.6, in most cases there is the possibility to split up an inference problem into a few independent tasks. Thus, most software packages like Stan or Turing.jl provide means to use multiple threads on a CPU for processing.

GPUs

The underlying frameworks of PyMC3 and TensorFlow Probability technically allow to compile code for GPUs, which offer many more threads than a typical CPU. However, the inference problems are often not structured in way that allows to exploit the advantage of GPUs easily, such that users of that packages usually use multiple threads on a CPU as well.

Cluster computing

For processing amounts of data that do not fit a single machine in terms of memory or compute power, computer clusters are used with dedicated frameworks like Spark [211] that provide an abstract API for running programs on a large-scale cluster. In the particular case of Spark, the code is executed in Java virtual machines (JVMs) that are hosted on the nodes of the cluster. The package Rainier [30] addresses this and provides code optimized for the JVM, thus allowing for efficient use on a cluster with spark. Note, however, that this way of parallelization only allows to distribute many similar inference problems on a cluster, not necessarily a single inference problem over several workers in a cluster.

6. Extending Turing.jl for online updates

In contrast to chapter 5 we turn our attention to a specific library for probabilistic programming and how to extend it for a specific application.

The results presented here are described in a manuscript which has been under review by the Journal of Statistical Software and will be slightly revised and resubmitted shortly after the writing of this thesis:

> K. Bob, B. Schmidt, and A. Hildebrandt. "Extending Turing.jl for Online Updates of Bayesian Inference in Julia"

The manuscript is directly replicated here with minimal changes. K. Bob contributed to the design of the package and writing of the article, implemented the software package and performed the case studies.

NUTS in online settings

As shown in chapter 3, the HMC algorithms and especially NUTS have many desirable properties for one-pass inference on data sets. However, these benefits come at the price of a high demand on computational resources, as for each proposed parameter sample NUTS needs to evaluate the product of prior and likelihood and its derivative several times. Especially in online settings, where small amounts of new data arrive steadily, simply re-running inference after each new datum can be prohibitively costly and would ignore previously gained information on model parameters.

SMC in online settings

An alternative approach is offered by the class of particle based inference methods ([51], [34]) which were also introduced in chapter 3. Instead of forming Markov chains, the distribution of interest is represented by a set of particles, i.e. coordinates in parameters space and their weights. Inference is done by adjusting coordinates and weights accordingly. Thus, particle based methods are better suited for efficient updates, as it usually requires only a simpler resampling, compared to a more costly rerun of NUTS. However, for high-dimensional or complex geometries of the parameter space, the simpler resampling often does not provide sufficient quality of exploration.

Combination

Taking the benefits and limitations of both the HMC and particle based methods together, it would be ideal to combine them such that the best from both worlds is available. This motivates our work to enable interchangeable usage of particle based methods for updating of parameter estimates that may have been inferred with another type of algorithm, especially NUTS.

Figure 6.1.: Inference architecture in Turing.jl. While usually a single object holds both the observed data x and the definition of the model $\mathcal{M}$, the sampler is an independent object. The inferred parameter sample $\{\theta\}$ is stored in a separate MCMC chain object.

6.1. The Turing.jl package as a basis

Unfortunately, to the best of our knowledge, popular software packages like Stan ([37]), PyMC3 ([176]) or the *Turing.jl* package by [71] do not provide the desired functionality yet.

Benefits

However, the *Turing.jl* package by [71] offers a perfect starting point for development of this feature, as it provides both particle based methods and MCMC algorithms like NUTS. Furthermore, it is designed in a modular fashion which allows for extensibility. As *Turing.jl* is written in the *Julia* programming language ([20]), we can immediately take advantage of its versatile feature set to create a highly efficient and scalable implementation. Moreover, Carpenter [36] mentions a great flexibility in model definition and a more flexible version of automatic differentiation as advantages of *Turing.jl* compared to other popular packages.

6.2. Technical description of the online update process

Inference architecture

The general architecture of a program for Bayesian inference in the Turing.jl package is shown in Figure 6.1, and the corresponding source code in the first block of Listing 6.1. Given observed data x and a model $\mathcal{M}$, a single object for holding both is used. In the next step, an appropriate sampler is chosen and constructed as an independent object. The inferred parameter sample $\{\theta\}$ from the run of the sampler is stored in a separate MCMC chain object.

Online update

If new data x' arrives, there are two steps necessary for updating the inference. First, a new model $\mathcal{M}'$ is needed, holding the new data and possibly new parameters as well. The second step requires to update the parameter samples. Instead of providing a new class of samplers for that, the new functionality is provided by converting an existing chain object with information from the new model into a particle container object, on which sequential samplers can operate, and converting back the resulting updated particle container into a chain object, see Figure 6.2 for a diagram of the online update process.

Chains and Varinfo

The two main building blocks for the process are `Chains` objects from the

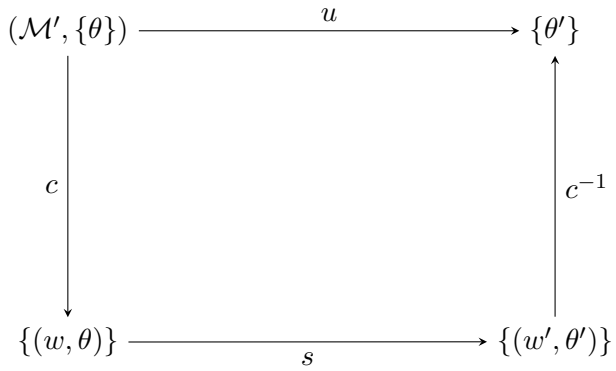


Figure 6.2.: Online updating of a MCMC chain $\{\theta\}$ given a new model $\mathcal{M}'$ by conversion of c to a particle container $\{(w, \theta)\}$, application of a sequential algorithm s resulting in an updated particle container $\{(w', \theta')\}$ and conversion back c^{-1} to an updated MCMC chain $\{\theta'\}$.

MCMCChains package as the representation of a chain $\{\theta\}$ and `VarInfo` objects that store the particle coordinates and weights in a particle container. Now, in the following, the single steps of the update process will be described in more detail.

Conversion to particle container

In the first step, a particle container is created from a given `Chains` object with the particle coordinates given by the parameters samples from the chain. If there are new parameters present in $\mathcal{M}'$, such that there are no samples for them in the chain, prior samples from $\mathcal{M}'$ are drawn and used as particle coordinates.

Sequential sampling

Using a particle based sampler for the inference is the second step. As Turing.jl provides a certain interface for the sampler, there is the possibility to choose from a large class of possible samplers.

Conversion from particle container

Finally, the particle coordinates and weights from the `VarInfo` objects in the particle container are extracted and written into the `Chains` object.

The whole process can be accelerated by using multiple threads in parallel.

We provided this functionality in a package *TuringOnline.jl*, which is available under https://github.com/hildebrandtlab/TuringOnline.jl

6.3. Usage example

Model definition

Listing 6.1 shows a code listing for the update process: we start with importing the packages that are needed for Bayesian modeling in *Turing.jl* (lines 1-3) and proceed by declaring the model from the 'Covid' use case, see Section 6.4 for details, (lines 5-29) and loading data into it (lines 31-33).

Sampling

Now were are able to sample from it, e.g. by using NUTS (line 35). If new data arrives (lines 38-39), we update the model (line 40) and we can perform the update by means of two lines of code (lines 42-43).

Table 6.1.: Overview of case studies. For each use case a short name, the type of model, the number of free model parameters and the number of data points is shown.

Name	Description	# Parameters	# Data points
Stocks	GARCH	4	4
Sensor	Gaussian process	3	40
Covid	Poisson regression	29	100

Ease of use

Missing parameters are added automatically. In this example we used the SMC algorithm from *TuringOnline.jl*, but it could be any other algorithm of type `ParticleInference`. Multi-threading is automatically available as well.

6.4. Case studies

In order to assess the performance of our software package in practice, we conducted three case studies in which we compared the computational cost and inference accuracy of our implementation to the implementation of NUTS in *Turing.jl*. The use cases were chosen to demonstrate both the broad applicability of our package in different fields and to examine different model complexities, see Table 6.1 for an overview. Concretely, we inspected the three following cases:

6.4.1. Use cases

'Stocks' use case

First, we studied a generalized autoregressive conditional heteroskedasticity (GARCH) model ([26]) for modelling of stock market returns, which will be referred to as the 'Stocks' use case. We assume the setting that for each point at time i the stock return y_i was observed. The mathematical formulation of the GARCH (1,1) model can be described as

$$\omega, \alpha, \beta \sim \mathcal{HN}(1) \tag{6.1}$$

$$\sigma_i^2 = \omega + \alpha\, y_{i-1}^2 + \beta\, \sigma_{i-1}^2 \tag{6.2}$$

$$y_i \sim \mathcal{N}(0, \sigma_i), \tag{6.3}$$

where $\mathcal{HN}(\sigma)$ denotes the Half-Gaussian distribution with scale parameter σ and again $\mathcal{N}(\mu, \sigma)$ the Gaussian distribution with mean μ and standard deviation σ. For T time points, there are 4 independent parameters for T observations. We chose $T = 4$ to generate a use case with a simple model and low computational cost.

'Sensor' use case

Second, we studied a Gaussian process (GP) model [171] for modeling of sensor data, which will be referred to as 'Sensor' use case. We assume the setting that for

Listing 6.1: Complete pass inference and online update in Turing.jl. The first part shows a complete pass inference in Turing.jl, the second part how to use the update module. See text in section 6.3 for details.

```
import Random as rnd
import Distributions as dist
import Turing as tur

tur.@model matrix_galm_poisson(cases,totals::Matrix{Int64}) = begin
    nt=size(cases,1)
    ns=size(cases,2)

    # Hyperpriors
    sigma ~ dist.Exponential(1)
    sigmaT ~ dist.Exponential(1)
    sigmaS ~ dist.Exponential(1)

    # latent space
    theta0 ~ dist.Normal(0.0,sigma)
    bTime ~ tur.filldist(dist.Normal(0.0,sigmaT), nt,1)
    bSpace ~ tur.filldist(dist.Normal(0.0,sigmaS), 1,ns)
    theta = logistic.(theta0 .+ bTime .+ bSpace)

    # likelihood
    for i = 1:nt
        for j = 1:ns
            cases[i,j] ~ dist.Poisson(totals[i,j]*theta[i,j])
        end
    end

    return (theta=theta,)
end;

cases_t = ...
totals_t = ...
model_t = matrix_galm_poisson(cases_t,totals_t)

chain_t = tur.sample(model_t, tur.NUTS(),...);

# Update with new data
cases_t_inc = ...
totals_t_inc = ...
model_t_inc = matrix_galm_poisson(cases_t_inc,totals_t_inc)

import online as on
chain_t_inc = on.update(chain_t,model_t_inc,tur.SMC(),rng)
```

each time point t_i the sensor records the signal y_i and a distance matrix D with entries $d_{ij} = d(t_i, t_j)$ is precomputed. The mathematical formulation of the model can be described as

$$\sigma \sim \mathcal{LN}(0, 1) \tag{6.4}$$
$$\phi \sim \mathcal{IG}(3, 0.5) \tag{6.5}$$
$$\mu \sim \mathcal{N}(0, 1) \tag{6.6}$$
$$\Sigma = \exp\left(-\frac{D^2}{\phi^2}\right) + \sigma^2 I \tag{6.7}$$
$$y_i \sim \mathcal{N}(\mu, \Sigma) \tag{6.8}$$

where $\mathcal{LN}(\mu, \sigma)$ denotes the lognormal distribution with location parameter μ and scale parameter σ, $\mathcal{IG}(\alpha, \beta)$ the inverse-gamma distribution with shape parameter α and scale parameter β. $\mathcal{N}(\mu, \sigma)$ is the Gaussian distribution with mean μ and standard deviation σ and I the identity matrix. For T time points, there are 3 independent parameters for T observations. Here we chose $T = 40$ to generate a case with a medium model complexity but larger computational cost.

'Covid' use case

Finally, we studied a hierarchical generalized linear model (GLM) with a poisson distributed response variable for epidemiological modelling, which will be referred to as 'Covid' use case. The model is a slight variation of the one by [202]. We assume the setting that for each time point i in each region j there are c_{ij} many infections observed among the population of size n_{ij}. The mathematical formulation of the model can be given by

$$\sigma, \sigma_T, \sigma_S \sim \mathcal{E}(1) \tag{6.9}$$
$$\theta_0 \sim \mathcal{N}(0, \sigma) \tag{6.10}$$
$$b_i^T \sim \mathcal{N}(0, \sigma_T) \tag{6.11}$$
$$b_j^S \sim \mathcal{N}(0, \sigma_S) \tag{6.12}$$
$$\theta_{ij} = \text{logistic}(\theta_0 + b_i^T + b_j^S) \tag{6.13}$$
$$c_{ij} \sim \mathcal{P}(n_{ij}\, \theta_{ij}) \tag{6.14}$$

where $\mathcal{E}(\lambda)$ denotes the exponential distribution with mean λ^{-1}, $\mathcal{N}(\mu, \sigma)$ the Gaussian distribution with mean μ and standard deviation σ, $\text{logistic}(\cdot) = \frac{1}{2}(1 + \tanh\left(\frac{\cdot}{2}\right))$ the logistic function and $\mathcal{P}(\lambda)$ the Poisson distribution with mean λ. For T points in time and S regions under consideration, there are $4 + T + S$ independent parameters for $T \cdot S$ observations. Here we chose $T = 20$ and $S = 5$ to generate a case with a more challenging model and high computational load. In each use case we studied both updating the model estimate for the last observation, i.e. from $T - 1$ to T (single update) and repeated updates for each $t \leq T$.

	Memory (GiB)		Runtime (s)		Score	
	NUTS	Online	NUTS	Online	NUTS	Online
Stocks	0.26	0.68	0.268	0.734	3.275	2.144
Sensor	6.65	0.53	4.382	0.766	7.745	7.864
Covid	8.44	3.03	7.061	3.359	-3.596	-5.418

Table 6.2.: Results for single update steps in the case studies. For each use case the memory estimate (lower is better), the median runtime in seconds (lower is better) and the score from equation 6.15 (higher is better) is shown.

6.4.2. Setup

Data used

We used synthetic data from prior predictive sampling in all use cases. This allows for assessing software performance without interference from the challenge of modeling a given data set accordingly. On real data it is often unclear which model to use and poor model choice is often reflected in poor computational performance (folklore theorem[1]), spoiling the comparison of different inference methods.

Computational environment

The study was performed on a single Ubuntu 18.04 LTS machine with a Intel® Cor i5-5300U CPU with 2 physical cores @ 2.30GHz, enabled hyper-threading allows for the parallel execution of 4 threads. The machine is further equipped with 16 GiB (2x 8 GiB) of SODIMM DDR3 Synchronous @1600 MHz main memory. We used Julia in version 1.6.2 and Turing.jl in version 0.17.4.

Benchmarking

For assessment of the computational cost of single updates we used *Benchmark.jl* [172] to measure the memory estimate and the median runtime among several samples.

Scoring

Gneiting and Raftery [79] give a detailed discussion for the assessment of the quality of a prediction P. Following their suggestions, we used the score

$$S(P, x) = -\log(\det \Sigma_P) - (x - \mu_P)^\top \Sigma_P^{-1} (x - \mu_P), \tag{6.15}$$

where x denotes the true value, μ_P the mean and Σ_P the covariance matrix of a posterior parameter sample P. The resulting values for single updates in each use case are shown in Table 6.2.

6.4.3. Results

'Stocks'

We found that for the lightweight problem in the 'Stocks' case the NUTS implementation both needs less computational resources and provides better inference quality. This sounds reasonable given the fact that there is an overhead for our update through the conversion of the chains and due to slightly worse exploration of the flatter likelihood landscape resulting from few data points.

[1] https://statmodeling.stat.columbia.edu/2008/05/13/the_folk_theore/

'Sensor' In contrast, for the 'Sensor' case the online implementation needs considerably less computational resources and even provides a slightly better inference quality. Following the above arguments, the computational load from the actual inference computations is higher compared to the overhead of converting the chains, such that the benefit of the simpler sampling comes into play. At the same time the model structure and amount of data allows for a convincing exploration of the parameter space by the simpler algorithm as well.

'Covid' In the 'Covid' use case, we found a mixed result: as the demand for computational resources is even higher, the savings in computational cost are significant again, but we see that the inference quality starts to degenerate as well, showing that the simpler sampling algorithm is challenged by the more complex model.

Repeated updates For the repeated updates setting we compared the inference quality using the score from (6.15) for a complete pass of all data up to t with NUTS, a complete pass with the particle based algorithm SMC from *Turing.jl*, which is used for the online steps, and once for repeated online updates.

Conclusion Overall, the findings are similar: in the 'Stocks' case the usage is of no benefit, as the NUTS implementation can deal with a complete inference for each $t \leq T$ easily. For the 'Sensor' case however, even repeated updates provide comparable results, see Figure 6.3. In the 'Covid' use case, we observed the problem that the results started to degenerate after repeated updates, a known behavior for particle based methods, see [51] for details. As a practical way to deal with it, we recommend a hybrid approach similar to the concept behind a Kalman filter: the online update mechanism provides updates for only a few steps while in the background, a more expensive computation like e.g. by a NUTS implementation is provided, such that the online implementation can periodically be 'guided back onto the right track'.

Based on our findings, we can conclude that for problems like 'Sensor' with medium model complexity but high demand for computational resources, the online package is best suited while for high-complexity models, it provides a fast update procedure that should be combined with additional steps to control model quality.

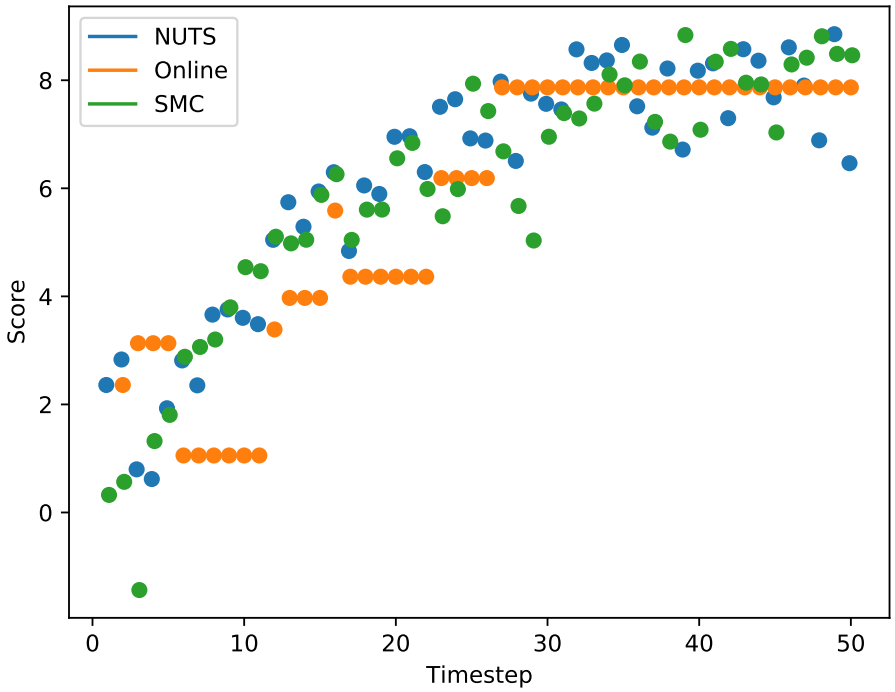


Figure 6.3.: Repeated updates for the 'Sensor' use case. For each time point t the score from (6.15) is computed for a complete pass of all data up to t with NUTS, a complete pass with the particle based algorithm SMC, which is used for the online steps, and once for repeated online updates. All approaches yield comparable inference quality. See section 6.3 for details.

Part II.

Applications

7. Regression analysis of surface parameters to quantify effect strength under uncertainty

While the previous chapter described the benefits of the Bayesian approach theoretically, this chapter presents a practical application to illustrate the typical steps in a Bayesian workflow and to give an example for a common type of model. Furthermore, we present action-guiding findings in the field of surface topography analysis.

The results presented here have been previously published in the following works:

> I. Calandra, L. Schunk, K. Bob, W. Gneisinger, A. Pedergnana, E. Paixao, A. Hildebrandt, and J. Marreiros. "The effect of numerical aperture on quantitative use-wear studies and its implication on reproducibility". In: *Scientific Reports* 9.1 (2019). ISSN: 20452322. DOI: 10.1038/s41598-019-42713-w

> A. Pedergnana, I. Calandra, K. Bob, W. Gneisinger, E. Paixão, L. Schunk, A. Hildebrandt, and J. Marreiros. "Evaluating the microscopic effect of brushing stone tools as a cleaning procedure". In: *Quaternary International* (July 2020). ISSN: 10406182. DOI: 10.1016/j.quaint.2020.06.031

> I. Calandra, K. Bob, G. Merceron, F. Blateyron, A. Hildebrandt, E. Schulz-Kornas, A. Souron, and D. E. Winkler. "Dental microwear texture analysis in Toothfrax and MountainsMap SSFA module: Different software packages, different results? [pre-print]". In: (Apr. 2021). DOI: 10.5281/ZENODO.4671439

For each publication K. Bob designed and implemented the Bayesian modeling and contributed to the writing of the article.

7.1. Surface topography analysis

The availability of various imaging technologies allows to analyze surface topographies at different scales [29]. In particluar the field of traceology studies wear marks and often serves research in archeology or criminology [197]. Examples in archeology are use-wear analysis of tools used by ancient humans [123] or dental wear [81] analysis to infer diets of animals of humans.

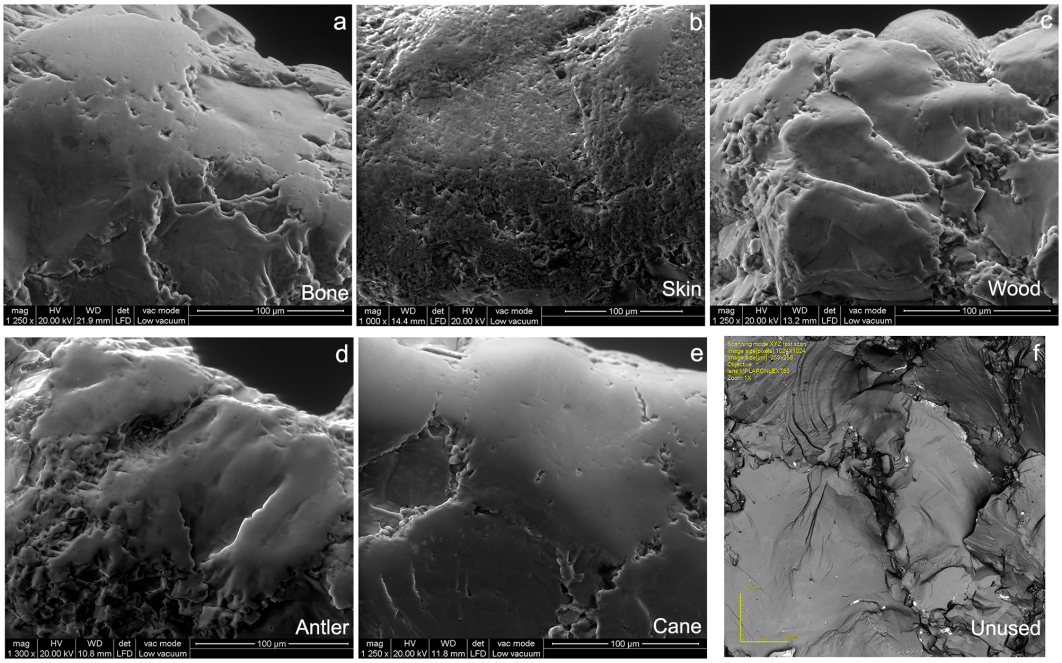


Figure 7.1.: Scanning electron microscope (SEM) images of a polished areas on quartzite flakes after contact with different materials (a-e) and LSM-maximum intensity image of an unused surface (f). Image taken from [158, Fig. 6], see there for further information.

As an example, Figure 7.1 shows the resulting wear marks on quartzite flakes of different contact materials. On a pure qualitative basis one can already observe the different characteristics of each contact material and conjecture that it might be even possible to distinguish them given their wear patterns.

Surface parameters

Favourably, in the last two decades the analyses of surfaces have become more quantitative, measuring different types of surface parameters, of which many are standardized [99]. Thus, it is possible to approach questions like the aforementioned one of distinguishability on a quantitative basis. While Table 7.1 gives an overview on those standardized parameters, Figure 7.2 illustrates some of those surface parameters.

Experiments

Ultimately many studies aim to infer traces on a sample with unknown wear, e.g. to find out the diet of an animal based on the wear of the teeth or the usage of a stone tool based on its wear marks. It is therefore necessary to have references with known wear. Accordingly, we conducted controlled experiments with known wear, which additionally allow to study the process of wear inference itself [32] [157] [33].

Data

In those experiments, we were interested in the relationship between an independent variable X (e.g. raw material, experimental setting, treatment) and a dependent variable Y, i.e. the surface parameters, for which pairs of observations (x, y) were recorded.

Table 7.1.: Overview on surface parameters after ISO 25178-2 standard [99]. Table taken from [156], see there for detailed discussion.

Type of Parameters	Examples	Functional Importance
Amplitude	*Sa, Sq, Sz, Sp, Sv*	Surface contact, lubrication, friction, wear, fatigue, technical control of manufacturing
Characterising the shape of the height distribution	*Ssk, Sku, Sp/Sz, Sq/Sa*	Surface contact, friction, wear
Spatial	*Sal, Str*	Lubrication, friction
Hybrid	*Sdq, Sdr*	Surface contact, friction, wear, ability to adhesive junctions, sealing, and cosmetic appearance
Related to material ratio curve	*Sk, Spk, Svk, Sr1, Sr2, Spq, Svk, Smq, Vvv, Vmp, Vmc, Vvc, Smc, Smr, Sxp*	Wear, friction, oil capacity, low wear assessment, technical control of manufacturing

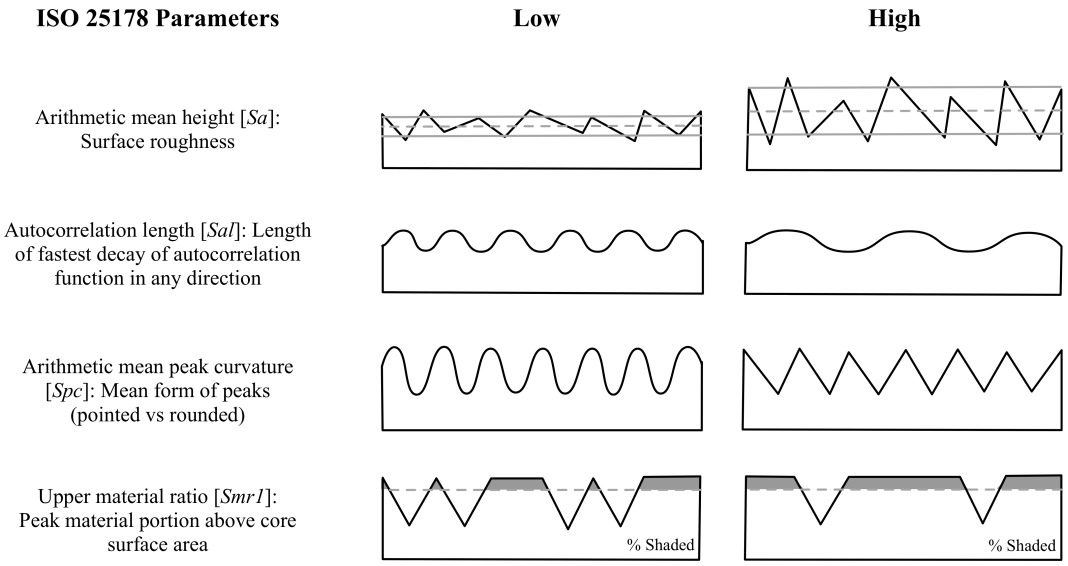


Figure 7.2.: Illustrations of some surface parameters. In each row there is a short description of a surface parameter and a sketch of a surface profile that would result in a low and high value of this parameter. Image taken from [124, Fig. 4].

7.2. Analysis by regression models

For each of the aforementioned studies statistical regression analysis was performed. As the analyses followed the same principles, but differed slightly each time, we will discuss the approach in general before turning to concrete results.

Model types

Regression models are often classified as linear or nonlinear in the independent variable. While the former are easier to treat, but sometimes too simplistic, the latter easily pose the danger of choosing an overly complicated model, hindering interpretation and treatment. Here, the family of generalized linear models [143] and derived versions strike a balance between the two extremes. Consequently, applications of these models spans various fields from finance [65] over epidemiology, see chapter 8 and chapter 9, to surface topography analysis here.

7.2.1. Model structure

Main components

In our analysis the relationship between the variables $Y = f(X)$ is modeled around two building blocks. The first one is

$$\mu(X) = \beta_0 + \sum_{i=1}^{n} \beta_i \, X_i + \sum_{i,j=1}^{n} X_i \, M_{i,j} \, X_j, \tag{7.1}$$

where X_i denotes different components of X, and β_0, β_i and the matrix $\mathbf{M}$ with entries $M_{i,j}$ are model parameters, such that θ contains at least $(\beta_0, \beta_i, M_{i,j})$.

The second building block relates $\mu(X)$ and Y by

$$Y \sim \mathcal{D}(\mu(X), \Theta_Y), \tag{7.2}$$

where $\mathcal{D}$ denotes a general probability distribution with (potential) other parameters Θ_Y. A common choice is $\mathcal{D} = \mathcal{N}$, the Gaussian distribution.

Complete model

Of course for Bayesian modeling priors for all model parameters are needed, such that a complete model specification would be

$$\beta_0 \sim \mathcal{D}_{\beta_0} \tag{7.3}$$

$$\beta_i \sim \mathcal{D}_{\beta_i} \tag{7.4}$$

$$M_{i,j} \sim \mathcal{D}_{M_{i,j}} \tag{7.5}$$

$$\mu(X) = \beta_0 + \sum_{i=1}^{n} \beta_i X_i + \sum_{i,j=1}^{n} X_i M_{i,j} X_j \tag{7.6}$$

$$\Theta_Y \sim \mathcal{D}_{\Theta_Y} \tag{7.7}$$

$$Y \sim \mathcal{D}(\mu(X), \Theta_Y) \tag{7.8}$$

Interpretation

Coming back to the distinction between linear and nonlinear regression models and the position of the generalized models within that spectrum, the structure in Equation 7.1 can be interpreted accordingly. Similar to approximating a complicated, but sufficiently well behaved function f by series expansion in contributions of certain order, Equation 7.1 tries to decompose the relationship $Y = f(X)$ into an overall level β_0, then effects that depend on the single components of the independent variable X, measured by the β_i and finally effects that depend on interactions between components of the independent variable, measured by the $M_{i,j}$.

Contrasts

If one is now interested in the difference in effect strength between different settings of X, the posterior distribution of the difference $\beta_i - \beta_j$, called *contrasts*, provides valuable information. Consequently, reported results are often contrasts. Figure 7.3 shows a typical result of one analysis: after the effect strength was attributed to the single factors (and interaction terms potentially), the contrasts between factor components can be computed and used to answer the questions at hand. If zero effect strength is not contained in the interval containing 95% of the estimated effect strength distribution, the effect is considered as significant.

Delivery to audience

As shown in Figure 2.1, the discussion of a statistical analysis can take place both at the model stage, that is in the space where the parameters Θ reside, and at the data stage, where given data X and predicted data $\tilde{X}$ reside. When the target audience is not very familiar with the statistical modeling, it helps to report the findings on the data stage, see Figure 7.4, such that the output can be understood straightforwardly.

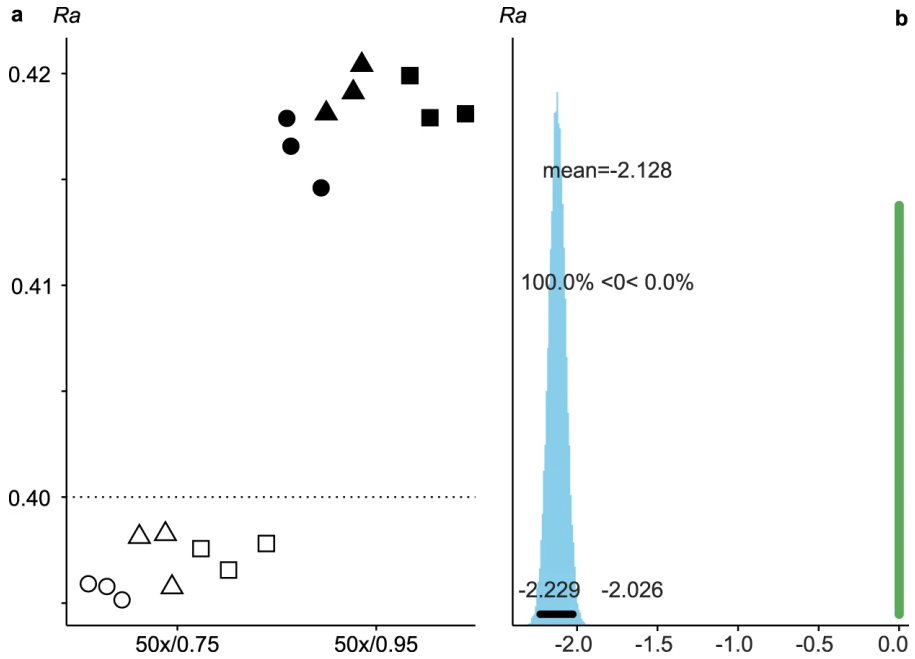


Figure 7.3.: Experimental data and estimated effect strength.
The left side shows experimental data for the surface parameter *Ra* in units of 1 µm, for different locations on the surface (factor 1, indicated by symbol shape) and different numerical apertures (factor 2, indicated by symbol color). The dashed line indicates the nominal value of the parameter on the roughness standard on which the measurements were taken.
The right side shows the effect strength attributed to change in numerical aperture (factor 2) by model A in Table 7.2 in units of 0.01 µm. The labels in the plot show, from top to bottom, the mean effect strength, the cumulative probability of all values smaller and bigger than zero effect strength and the bounds of the interval containing 95% of the probability distribution. The vertical green bar marks zero effect strength. Image taken from [32, Fig. 3].

7.2.2. Choice of model components

The selection of the prior distributions $\mathcal{D}_{\beta_0}, \mathcal{D}_{\beta_i}, \mathcal{D}_{M_{i,j}}$ and the likelihood $\mathcal{D}_{\Theta_Y}$ requires almost always two choices: the respective distribution itself and the parameters of this distribution.

From simple to hierarchical models

In the basic case, the parameters of the distributions are fixed, e.g. $\mathcal{N}(0,1)$. However, due to the numerical treatment of the models it is also possible to use distributions whose parameters are themselves random variables, which allows to express uncertainty in a more nuanced manner. In that contexts the distributions describing the parameters of the priors are called *hyperpriors* [74, Section 5.2]. On top of that, by letting different parameters share a common prior or hyperprior, one is using the advantageous hierarchical models. These allow to propagate information between parameters, see e.g. section 3.4.3 for an example and further discussion.

Maximum entropy principle

One possible guide for the selection of distributions is the *maximum entropy principle* [102] [103] [101, chapter 11] [127, Section 10.3]. It states that one should chose the prior distribution that maximizes the Shannon entropy [184] under the constraints given by the prior knowledge. Park and Bera [155, Table 1] give, amongst others, the following examples: while a parameter with no moment constraints and finite support should be modeled by the uniform distribution, a parameter with infinite support and constraints on the first and second moment should be modeled by a Gaussian distribution.

Gelman et al. [74, Sections 2.4 and 2.6] and Gelman et al. [75, Section 7.3] follow a different approach especially for prior choice and distinguish classes of priors based on the amount of information contained.

For a different perspective, see the work of Snoussi [187] who gives a guidance on prior selection from a information geometry point of view.

Heavy tails

When choosing likelihoods, information on the measurement process should be taken into account, potentially using the maximum entropy principle again. However, as Taleb [193] points out, it is especially important to consider whether the distributions need to model heavy tails.

Validation

As discussed in section 2.6, comparing data with the prior predictive distribution (2.5) is a meaningful way to access the validity of the model component choice. Figure 7.4 shows an example, where both prior and posterior predictive distributions and the experimental data are shown together.

7.3. Studies conducted and resulting findings

While in the previous sections a class of models was more abstractly discussed, this sections puts the focus back onto the surface topography analysis.

Goals of modeling

The main goal of the modeling was to attribute the strength in effects to the different components separately and thus being able to compare different settings within a component. Moreover, the model enables to predict the outcome in settings

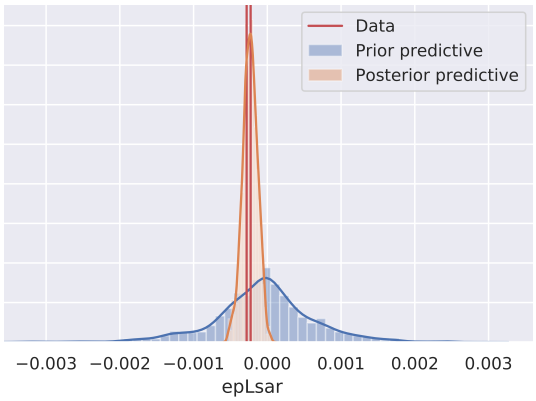


Figure 7.4.: Prior and posterior predictive distribution.
Prior and posterior predictive distribution and experimental data for a surface parameter studied in [157] are shown. As desired, the prior predictive distribution is wide and unbiased, whereas the posterior predictive distribution is centered around the data and considerably smaller. Thus we can conclude that the prior chosen is not too strong and the information from the data is well absorbed into the posterior distribution.

that were not measured. A strong benefit of the Bayesian approach to the model parameter inference is that we can quantify the uncertainty of our findings as well.

Studies conducted

In total, 3 different studies were conducted: first, in [32] we investigated the effect of the numerical aperture on the surface parameter measurements in laser-scanning confocal microscopy with two objectives having the same magnification but different numerical apertures.

Second, in [157] we performed controlled experiments to evaluate the microscopic effect in terms of change of surface parameters of brushing as a cleaning procedure.

Finally, in [33] we investigated the effect of using the same workflow in different software packages for processing the surface topography data on the computed values of the surface parameters.

Individual models

Although their overall structure is similar, the models used in the previously mentioned studies differ in details. Table 7.2 shows the specification of the individual models. Figure 7.5 shows the structure of a single model as an example. It is a hierarchical model with 334 parameters but could due to that be inferred on 230 data points only, while still providing credible results.

Table 7.2.: Regression models for surface topography analysis.
The column names refer to equations (7.3) to (7.8). For a list of used probability distributions and their symbols see the list of symbols in the frontmatter. m denotes sample mean estimates, s, s_i, s_M sample standard deviation estimates and $e_{\max} = 0.2 \min s_i$. Cells with $-$ mean that the corresponding part is not used in the model. The symbol † indicates that for $i = 2$ the corresponding distribution of model E is used.

Model name	Reference	$\mathcal{D}_{\beta_0}$	n	$\mathcal{D}_{\beta_i}$	$\mathcal{D}_{M_{i,j}}$	$\mathcal{D}_{\Theta_Y}$	$\mathcal{D}$
A	[32]	$\mathcal{N}(m, s^2)$	2	$\mathcal{N}(0, s_i^2)$	$\mathcal{N}(0, s_M^2)$	$\epsilon \sim \mathcal{U}(0, e_{\max})$	$\mathcal{N}(\mu, \epsilon^2)$
B	[157]	$\sigma_0 \sim \mathcal{HN}(s_0)$ $\mathcal{N}(0, \sigma_0^2)$	2	$\sigma_i \sim \mathcal{HN}(s_i)$ $\mathcal{N}(0, \sigma_i^2)$	$\sigma_M \sim \mathcal{HN}(s_M)$ $\mathcal{N}(0, \sigma_M^2)$	$\epsilon \sim \mathcal{U}(0, e_{\max})$	$\mathcal{N}(\mu, \epsilon^2)$
C	[33]	$\sigma_0 \sim \mathcal{HN}(s_0)$ $\xi_0 \sim \mathcal{N}(0, 1)$ $\mathcal{N}(\xi_0, \sigma_0^2)$	3	$\sigma_i \sim \mathcal{HN}(s_i)$ $\xi_i \sim \mathcal{N}(0, 1)$ $\mathcal{N}(\xi_i, \sigma_i^2)$	–	$\nu \sim \mathcal{E}(\lambda_\nu)$ $\sigma \sim \mathcal{G}(\alpha_\sigma, \gamma_\sigma)$	$\mathcal{T}(\mu, \sigma, \nu)$
D	[33]	$\sigma_0 \sim \mathcal{HN}(s_0)$ $\xi_0 \sim \mathcal{N}(0, 1)$ $\mathcal{N}(\xi_0, \sigma_0^2)$	2	$\sigma_i \sim \mathcal{HN}(s_i)$ $\xi_i \sim \mathcal{N}(0, 1)$ † $\mathcal{N}(\xi_i, \sigma_i^2)$	$\sigma_M \sim \mathcal{HN}(s_M)$ $\xi_M \sim \mathcal{N}(0, 1)$ $\mathcal{N}(\xi_M, \sigma_M^2)$	$\nu \sim \mathcal{E}(\lambda_\nu)$ $\sigma \sim \mathcal{G}(\alpha_\sigma, \gamma_\sigma)$	$\mathcal{T}(\mu, \sigma, \nu)$
E	[33]	$\sigma_0 \sim \mathcal{HN}(s_0)$ $\xi_0 \sim \mathcal{N}(0, 1)$ $\mathcal{N}(\xi_0, \sigma_0^2)$	1	$\gamma_i \sim \mathcal{HN}\left(\frac{3\sqrt{6}}{2\pi} s_i\right)$ $\xi_i \sim \mathcal{N}(0, 1)$ $\mathfrak{G}(0, \gamma_i)$	–	$\nu \sim \mathcal{E}(\lambda_\nu)$ $\sigma \sim \mathcal{G}(\alpha_\sigma, \gamma_\sigma)$	$\mathcal{T}(\mu, \sigma, \nu)$

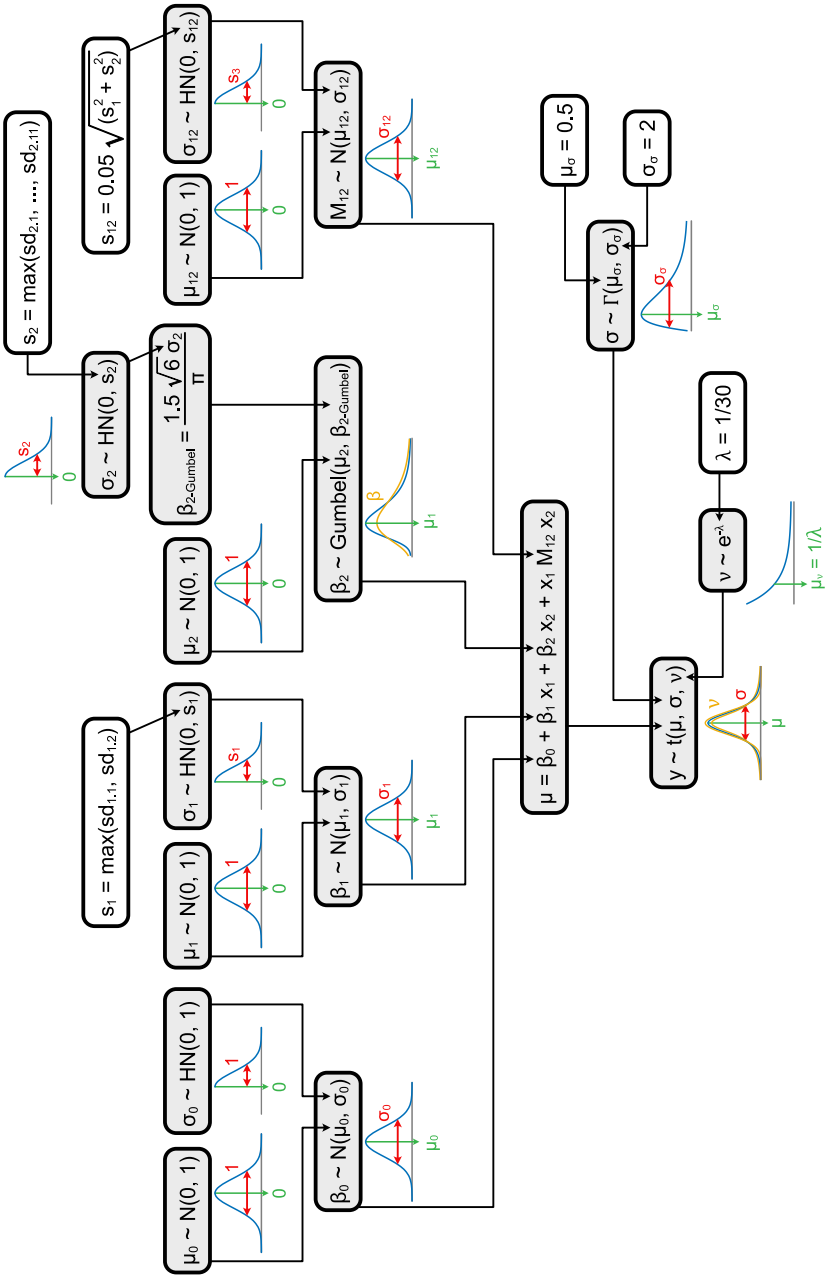


Figure 7.5.: Structure of the regression model D from Table 7.2.
The components of the model are shown as a tree starting at the likelihood. Arrows indicate dependency, grey boxes random variables and white boxes deterministic values. If random variables are drawn from a distribution, the probability density function of the corresponding distribution is sketched. The figure is slightly adapted from the supplementary material to the work of [33] and was originally created by Dr. Ivan Calandra.

Table 7.3.: Significant changes in surface parameters due to cleaning. Table taken from [157], see there for detailed discussion.

Surface parameter	Treatment		
	Rubbed, dirt applied	Brushed, no dirt applied	Brushed, dirt applied
epLsar	False	True	True
HAsfc9	False	True	False
Sku.SL	False	True	False
Smr.SL	True	False	False
Sxp.SF	False	True	False
Str.SF	False	True	False
Vvv.SF	True	True	False
Isotropy.SF	False	True	False
Isotropy.SF.1	False	True	False

Using the approaches described we could find the following results:

Effect of numerical aperture

First, in the study [32] flint and quartzite tools were imaged using laser-scanning confocal microscopy. By the regression model we could isolate the effect of changing the numerical aperture of the objective that is used for imaging, see Figure 7.6 for the results on selected surface parameters. We showed this has significant effects on almost all surface parameters under study, having strong implications for the reproducibility of a study. Consequently we recommended to report all settings of acquisition and analysis.

Effect of cleaning procedures

In [157] we investigated common cleaning procedures of artifacts. By the regression model we could isolate the effect from the cleaning procedures after measuring surfaces of different materials before and after the treatments. We found that the cleaning significantly changed nine surface parameters, see Table 7.3. Thus caution for future cleaning attempts is required not to possibly alter the functional interpretation of an artifact unintentionally.

Differences due to software

Finally, in [33] we showed that two different software packages for processing the surface topography data yielded significantly different outputs for all surface parameters, which were testable, making the values not directly comparable and preventing a merge of values from different software packages.

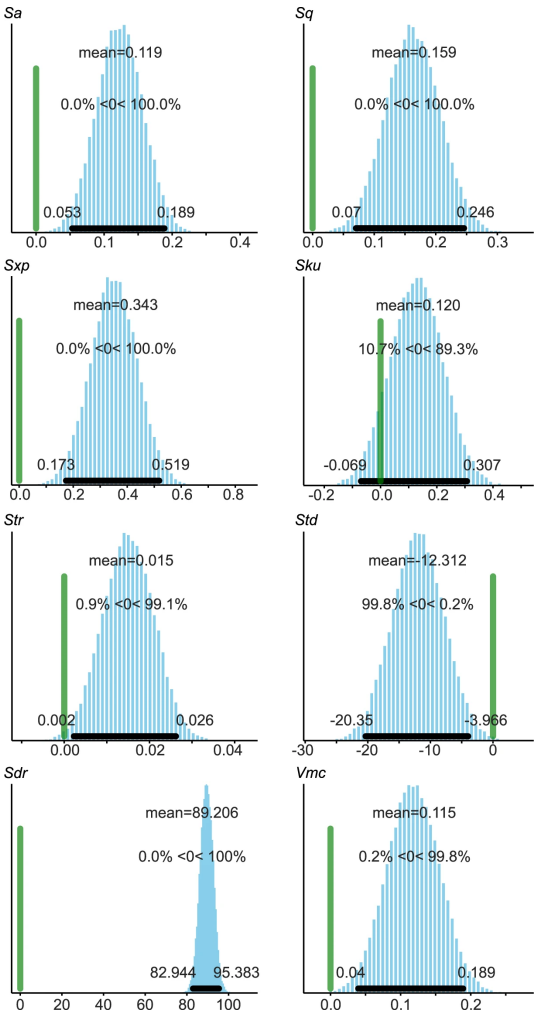


Figure 7.6.: Significant effect strength of change in numerical aperture.
Effect strength attributed to change in numerical aperture (factor 2) by model A in Table 7.2 for selected surface parameters. See caption of Figure 7.3 for a detailed description of the annotations in the subplots. Image taken from [32, Fig. 2].

8. Epidemiological modeling of infectious diseases

For the last part of this work we turn our attention to the application of Bayesian modeling in modeling the epidemiology of infectious diseases. We will start with a review of the field in this chapter.

8.1. Epidemiology of infectious diseases

While the field of epidemiology deals with the distribution health and all kinds of diseases in a population [182], we will focus here on the spread of infectious diseases, i.e. diseases that are caused by pathogenic organisms like bacteria and viruses.

Epi- and pandemics

As for an infectious disease, the number of cases may increase multiplicatively, it is possible that a large percentage of the population may be affected. Following [62], an *epidemic* occurs if an infectious disease spreads rapidly in short time in a population, and if the epidemic affects several continents, it is called a *pandemic*. Historic examples are, among others, the influenza-A-H1N1 pandemic lasting from 1918 to 1919 or the spread of HIV since the late 20th century, each having caused large number of deaths and severe conditions.

Public health

To prevent damage from infectious diseases and others, countermeasures and preventive actions are taken and studied in the field of public health. Cohen [42] describes that by means of better hygiene, availability of antibiotics and vaccinations, better quality of water, food and housing, in the 20th century infectious diseases decreased substantially and were even considered beaten. However, he continues, the factors of rising international travel and commerce, changes in demographics and behavior, environmental change and land use and microbial adaption and change could lead to increases in susceptibility and or transmission, the emergence of new disease, or the evolution of new or drug-resistant microorganisms.

8.2. Modeling infectious diseases in a population

Mathematical models are used for both scientific understanding and to aid development of interventions [80]. We will shortly describe three classes of models and end with a short discussion on parameter estimation.

8.2.1. Compartmental models

Compartmental models are class of models where the population is split into compartments indicating different statuses like being susceptible, infected or recovered etc. Typically, a homogeneous mixing of the population is assumed and the interactions are often deterministic.

SIR model

A simple compartmental model is the so-called SIR model, where three compartments model the susceptible S, infected I or removed R, i.e. immune or deceased parts of the population. The mathematical formulation of the model is the following system of coupled ordinary differential equations

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\frac{\beta IS}{N}, \tag{8.1}$$

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\beta IS}{N} - \gamma I, \tag{8.2}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I, \tag{8.3}$$

where $N = S + I + R$. Depending on initial conditions and time dependence of the parameters β and γ solutions of varying kinds of complexity exist, see [108] and [179] for an extended discussion.

Reproduction number

The behavior of the model is often discussed in terms of the *basic reproduction number* R_0 given by

$$R_0 = \frac{\beta}{\gamma}. \tag{8.4}$$

An important observation is that $R_0 > 1$ leads to phases of approximately exponential growth in I in a largely susceptible population, as an approximation of equation (8.2) shows:

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\beta IS}{N} - \gamma I \approx (\beta - \gamma)I, \tag{8.5}$$

when $S \approx N \gg I$ and thus

$$I(t) \approx I_0 \, \mathrm{e}^{(\beta-\gamma)t} \tag{8.6}$$

with $(\beta - \gamma) > 0$. Consequently, the reproduction number of infectious diseases is of great importance to the course of their spread.

Extensions

Derived models with different choices of compartments exists, depending on the modeled disease. Either further compartments are added for e.g. vaccinated parts of the population or the compartment R for removed is not considered, when parts of the population becomes susceptible again. Moreover, additional stochastic terms and time-dependence of the coupling parameters like β and γ are also considered, see e.g. [52] for further discussion.

COVID-19 related modeling

Dehning et al. [49] used a SIR model with time-dependent coupling parameter β to infer change points in the spread of COVID-19 and to quantifiy the effect of

interventions taken. Albani, Velho, and Zubelli [3] used an extended model with several compartments added.

Problems

The assumptions on which compartmental model are build are lacking two aspects [95]: first, the mean approximation breaks when only a small number of cases happend at the beginning of an outbreak and second, the mixing of population is often incomplete, sucht that a contact network or a transmission tree [80] gives a better description.

8.2.2. Network based models

Networks are recognized as a universal structure behind a large number of phenomena, see e.g. the work of Albert and Barabási [4]. Consequently, network based models are used in epidemiological modeling as well, see e.g. the works [95] [12] [28] that stress the importance of the network topology for the spreading behavior.

Network topology

The network topology used can be either from observations or from synthetic data using models for network topologies, see [104] for an extended discussion. Albert and Barabási [4] give an overview on different models for network typologies. Social networks, i.e. networks that describe phenomena that are governed by human interaction, typically have similar properties like a power law degree distribution and a small effective diameter, see [114] for further discussion.

Stochastic interactions

Typically, network based models are not tractable in closed form and thus analyzed numerically. Considering stochastic interactions instead of deterministic ones is then usually no further restriction, see e.g. [95].

COVID-19 related modeling

Network models were used in modeling the COVID-19 spread, see e.g. the work [39] that used mobile phone data to construct mobility networks in connection with a SEIR model, the work [68] that simulated control strategies in a network based on real-world data and the work [121] that used a SIR on synthetics networks.

8.2.3. Models for disease surveillance

Both classes of models discussed so far model the spread of infectious diseases over time and can be used for parameter inference and prediction of future events when applied to observational data. However, the necessary information may not be available at all or only with a significant delay.

Disease surveillance

This is of great consequence to the field of *disease surveillance*, that deals with the following tasks [148]: early detection of an emerging pandemic, early assessment of characteristics of the disease, monitoring and investigation of the effectiveness and impact of interventions. Consequently, additional models are needed to provide real time information, especially for decision making. We will now discuss the obstacles in more detail with an emphasis on the COVID-19 pandemic, although the requirements are to a large degree of a general nature.

Disease mapping

Best, Richardson, and Thomson [14] give an overview on spatial Bayesian mod-

els for disease mapping, that allows, e.g., to investigate correlations of rare diseases to other factors. A recent example is the work [177] that deals with the COVID-19 pandemic using a SEIR model with additional spatio-temporal information.

Imputation

Unknown infection dates

When dealing with case report data, often the date of infection T_I is unknown and only the date of registration T_R is known. Thus the date of infection needs to be imputed, see e.g. [83] for imputation during the COVID-19 pandemic. This work used a generalized additive model for location, scale and shape (GAMLSS) [188]

$$T_R - T_I \sim \mathcal{D}(\mu, \sigma, \tau, \dots), \tag{8.7}$$

where the distribution parameters $\mu, \sigma, \tau, \dots$ are related to explanatory variables x_i by

$$\eta_j = \beta_{0,j} + \sum_{i=1}^{n} \beta_{i,j}\, x_i \qquad \eta_j \in \{\mu, \sigma, \tau, \dots\}. \tag{8.8}$$

By sampling from an inferred distribution $\mathcal{D}$ dates of infection T_I are then imputed.

Nowcast

Reporting delay

The time delay between infection and registration or hospitalization of cases [57] creates the need for *nowcasting*, i.e. for computing the state of the spread now from past observations. Höhle and Heiden [91] developed an approach that was extended by Günther et al. [83] for the COVID-19 pandemic. Following Höhle and Heiden [91], the precise definition of the task is given as follows: let the difference $D \equiv T_R - T_I$ be defined as the delay D and $N_{T_I,D}$ denote the number of cases that occur on T_I and are registered at $T_R = T_I + D$. The task of nowcasting is now to estimate

$$N_{T_I} = \sum_{D=0}^{T-T_I} N_{T_I,D} \; + \sum_{D=T-T_I+1}^{\infty} N_{T_I,D} \tag{8.9}$$

on day $T > T_I$, given observations $\{n_{T_I,D}\}_{D=0}^{T-T_I}$. In equation (8.9), the first sum describes all cases that are already known at day T, such that the value of the sum is known from data. The second term, however, describes all cases that will be reported later than T and are thus unknown at T. Consequently, the key challenge is to model the contribution of the second term.

Maximum delay

Often, one assumes a maximum delay $D_{\max}$, such that the second term in (8.9) becomes a finite sum.

Delay distribution

One approach to nowcasting is modeling the delay distribution

$$P_{D|T_I}(d\,|t_I) = p_{d|t_I}. \tag{8.10}$$

See Höhle and Heiden [91] and Günther et al. [83] for various possibilities.

Another approach uses hidden Markov models, where the undelayed infections are considered in the latent state space and the delayed reports are considered as emissions, see the work of Nunes, Natário, and Lucília Carvalho [150] and Zou, Zhang, and Yan [215] for a complex spatio-temporal hidden Markov model. Hidden Markov Models

Vidal Rodeiro and Lawson [202] discuss a model formulation and sampling technique that allows for efficient online updates of data and model estimates. Birrell et al. [22] discuss online updates by SMC for modeling influenza outbreaks. Online updates

Forecast

Next to correcting the imperfections of the reporting system, a short-term forecast may required to aid planning. See Stojanović et al. [189] for a general approach to prediction of the spread of infectious diseases and Albani, Velho, and Zubelli [3] for a work related to the COVID-19 pandemic.

8.2.4. Evaluation of probabilistic assessments

When evaluating probabilistic assessments, there are several challenges [78], which require a set of proper scoring rules [79]. The work [27] has a special focus on epidemic forecasts.

8.2.5. Parameter estimation

Estimation of the parameters of epidemiological models from observational data is a key task. Many works that we have referred to so far [91][189][49][83][92] have used a Bayesian approach. Furthermore, Lawson [113] published a text book on Bayesian methods for health modeling, Nguyen et al. [147] discusses parameter estimation for ODEs with some comment on Bayesian methods and Holbrook et al. [92] provide a framework for Bayesian inference in the context of Big Data. Bayesian modeling

9. Nowcasting infection numbers

Having provided the necessary background in chapter 8, we are now ready to cover our contributions in this chapter.

The results presented here have been previously presented in the following work, which will be submitted shortly after the writing of this thesis:

> A.-K. Hildebrandt, K. Bob, D. Teschner, T. Kemmer, J. Leclaire, B. Schmidt, and A. Hildebrandt. "CorCast: A Distributed Architecture for Bayesian Epidemic Nowcasting and its Application to District-Level SARS-CoV-2 Infection Numbers in Germany". In: *medRxiv* (June 2021). doi: 10.1101/2021.06.02.21258209

and which will be directly replicated here with minimal changes. K. Bob contributed to the design of the system, the Bayesian modeling and writing of the article.

Due to the intended publication as a standalone manuscript, the concepts of imputation and nowcasting will be shortly explained again, although they were already discussed in subsection 8.2.3.

Furthermore, the methods for faster inference covered in chapter 4 and chapter 6 could be used here with potentially great benefit and were in fact developed due to the high computational effort required for this project. However, combining all approaches together is not straightforward and will be covered in future work.

9.1. Introduction

The SARS-CoV-2 pandemic that emerged in late 2019 has clearly shown the need for accurate, timely, and fine-grained infection statistics. To assess infection risks for different parts of the population, e.g., different age groups, the current number of daily infections for that sub-population in the geographic region of interest is obviously crucial. Corresponding datasets also allow for the estimation of the current replication number R, and thus enable a judgement about the efficiency of current anti-epidemic measures, such as social distancing or mandatory wearing of private protective equipment such as face masks.

Unfortunately, the data required for these applications is typically not directly available due to a number of problems of both fundamental and practical nature, some of which are exacerbated by the particular properties of SARS-CoV-2. To illustrate these problems, let us assume that a person p_i is infected at date x_i. After

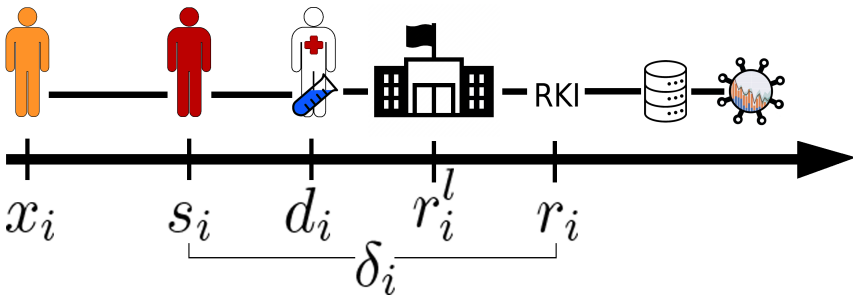


Figure 9.1.: Timeline of dates during infection and reporting. A person gets infected at date x_i, the disease begins at date s_i, is diagnosed at date d_i and gets registered at dates r_i^l at the local health authority and r_i at the RKI, Germany's public health institute. $\delta_i := r_i - s_i$ is called the reporting delay. After registration at RKI, the case is reported in a database and can be fetched by CorCast.

an incubation period, the disease begins at date s_i, is diagnosed at date d_i, and finally registered in a database at date r_i, together with relevant information about p_i (such as gender, age, and geographical region). The whole process is summarized in Figure 9.1.

Obviously, not all infections are identified. In fact, it is generally unknown, at least until detailed antibody studies can be performed, what the true infection rate is. This is particularly true for diseases such as SARS-CoV-2, which lead to a large percentage of asymptomatic or only mildly symptomatic cases. A further complication arises from the fact that, even for identified cases, the date of disease onset s_i is often unknown, for instance in asymptomatic cases that were identified through contact tracing or routine mass testing. In particular, it is not generally possible to assume that $s_i = d_i$, and in particular that $s_i = r_i$. Instead, each case has an individual *reporting delay* that is composed of the delay between disease onset and diagnosis as well as the delay between diagnosis and reporting, and this delay is unknown for a large percentage of the reported cases.

Owing to the reporting delay, information about infection numbers at the most recent days is necessarily inaccurate, or rather, incomplete. The goal of *nowcasting* of infection numbers is thus to infer an estimate of the true infection numbers based on the current, incomplete counts and the reported infection history [91]. This task is often decomposed into two phases [83]: the imputation of disease onsets for cases that have already been reported, but for which the true onset is unknown, and the nowcasting based on this data. These phases consist of a number of steps, including data ingestion, data cleanup and preprocessing, imputation of disease onset, nowcasting, postprocessing, validation, and reporting.

9.1.1. Our contribution

When developing a Bayesian model for imputation and nowcasting of the SARS-CoV-2 infection numbers in Germany at a district level, we encountered a number of challenges. For instance, each step or module should be individually exchangeable, which greatly simplifies development and debugging and allows for testing and validation of combinations of module variants. To ensure reproducibility, it is imperative that data, modules, and workflows can all be versioned in a manner that allows exact recreation of the conditions at a specific point in time. Due to the large amounts of data generated in the exponential phase of a pandemic, pipelines further need to scale out sufficiently. Finally, deployment and maintenance of the system needs to be addressed.

Despite their crucial importance, these practical steps of implementation, deployment, operation, and maintenance are often ignored in the literature.

Here, we report on a stable and scalable distributed system, called CorCast, for the reproducible estimation of nowcasts suitable for pandemic scenarios. CorCast's implementation is highly modular – indeed, retargeting CorCast to other geographical regions or other diseases can be achieved by adapting only the data ingest and dashboarding modules. Similarly, the influence of different preprocessing methods, different imputation models, or different nowcast models on the final nowcast can be systematically studied. By basing CorCast on the Pachyderm framework [149], we guarantee reproducibility. Scaling out is achieved through the use of big data concepts and a scalable Kubernetes deployment.

To validate our architecture, we build a full pipeline for the nowcasting of Covid-19 infections in Germany on a district, state, and country-wide level (cf. Figure 9.2).

9.1.2. Related work

Related work, such as Höhle and Heiden [91] followed, e.g., by Stojanović et al. [189] and Günther et al. [83], mainly focused on statistical modeling for nowcasting, while Dehning et al. [49] and Albani, Velho, and Zubelli [3] dealt also with forecasting. Software packages for disease monitoring are covered, e.g., by Salmon, Schumacher, and Höhle [174], Salmon et al. [175], and Höhle [90], but to the best of our knowledge, no general framework for the development of such prediction techniques has been proposed prior to this work.

9.2. Methods

9.2.1. Preprocessing

The RKI provides so-called "publications" that contain sufficient information to query the number of infections, recoveries, and deaths as a function of r_i^l, i.e., the

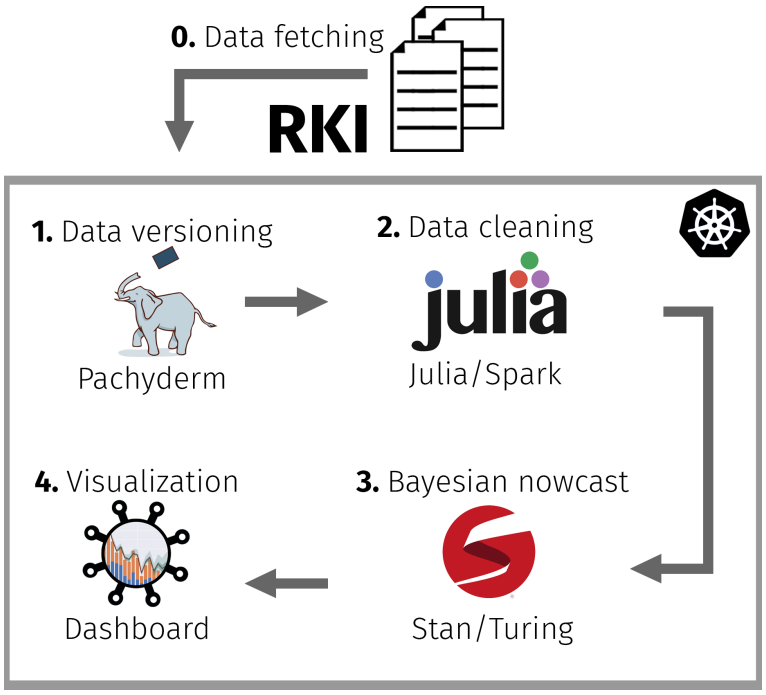


Figure 9.2.: Graphical representation of the nowcast workflow. Data is fetched from the Robert Koch Institute, Germany's public health institute, once per day (0). All versions are saved and checked in for tracking with Pachyderm (1). Data needs to be cleaned and standardized by a set of custom scripts written in the Julia programming language (2). Afterwards, imputation and nowcast can be performed (3). Lastly, processed data and nowcast results are visualized via a custom designed dashboard (4). All CorCast components are orchestrated through Kubernetes.

date at which they were recorded locally at the health department of the district. Unfortunately, the files do not contain the date at which they were recorded centrally by the RKI, r_i. Since much of the information provided by the RKI, such as imputed disease onsets and nowcasts, seems to be predicted based on r_i rather than on r_i^l, the lack of this information is rather unfortunate. To recreate r_i, it is necessary to keep an archive of all previous daily data publications of the RKI (a task for which Pachyderm is ideally suited), and iterate backwards to identify at what date a case first appeared in the RKI publications. Since the data publications also contain corrections, which are modelled as deletions of cases in previous publications and new insertions of these cases, recreating r_i is non-trivial and error-prone. In addition, the publications contain numerous inconsistencies and errors that have to be identified and corrected for as accurately as possible. Finally, the process is made considerably more cumbersome by the fact that the data format varies over time. Dates, in particular, are stored in at least four different formats, depending on the date of the publication and the data column. Hence, parsing and preprocessing the input data is relatively complex. At the same time, the preprocessing needs to be highly efficient: individual cases have to be tracked through hundreds of files, each of which needs several steps of data conversion, processing, and quality control. Coupled with the large case numbers of the pandemic, preprocessing becomes quite resource intensive.

The ingestion pipelines at the start of our Pachyderm workflows start with a module that downloads the current RKI data publication and archives it in a Pachyderm repository. In the next step, the newest publication is processed and converted into a dataframe, which is then serialized to another Pachyderm repository. Simultaneously, we attempt to reconstruct the missing r_i values through iteration over all previous data publications[1]. The results are again stored in a Pachyderm repository. This approach allows us to base further processing steps on either the RKI-provided r_i^l or on the missing r_i values.

9.2.2. Imputation models

Our baseline model is defined as

$$\mu \sim \mathcal{RN}_{[0.1,10]}(1, 2) \tag{9.1}$$

$$\sigma \sim \mathcal{RN}_{[0.1,10]}(1, 2) \tag{9.2}$$

$$\delta \sim \mathcal{LN}(\mu, \sigma), \tag{9.3}$$

where $\mathcal{RN}_{[l,u]}$ denotes a truncated normal distribution and the priors for μ and σ are only weakly informative and positive. The fully hierarchical model shown in Figure 9.4 turned out to be infeasibly costly to compute on the large amounts of data generated during the pandemic. As a less sophisticated variant, we use a manual

[1] The details of the implementation are rather complex to handle a variety of special cases.

approximation to the hierarchical model by first fitting a global model for the delay distribution to all cases in Germany, then a model with means and variances per state of the form

$$\mu_s \sim \mathcal{RN}_{[0.1,10]}(\mu_\mathrm{g}, 0.5), \quad 1 \leq s \leq 16 \tag{9.4}$$
$$\sigma_s \sim \mathcal{RN}_{[0.1,10]}(\sigma_\mathrm{g}, 0.5), \quad 1 \leq s \leq 16 \tag{9.5}$$
$$\delta_i \sim \mathcal{LN}(\mu_s, \sigma_s), \quad s = \mathrm{state}(\mathrm{case}(i)), 1 \leq i \leq \#cases \tag{9.6}$$

and, finally, fitting a more complex model with additional covariates (here: age group, gender, district, weekday, and number of weeks since the start of the pandemic):

$$\mu_d \sim \mathcal{RN}_{[0.1,10]}(\mu_s, 0.5), \quad s = \mathrm{state}(\mathrm{district}(d)), d \in \mathrm{districts} \tag{9.7}$$
$$\sigma_d \sim \mathcal{RN}_{[0.1,10]}(\sigma_s, 0.5), \quad s = \mathrm{state}(\mathrm{district}(d)), d \in \mathrm{districts} \tag{9.8}$$
$$\beta_{\mu,\mathrm{a}} \sim \mathcal{N}(0, 0.5), \quad a \in \text{age groups} \tag{9.9}$$
$$\beta_{\sigma,\mathrm{a}} \sim \mathcal{N}N(0, 0.5), \quad a \in \text{age groups} \tag{9.10}$$
$$\beta_{\mu,\mathrm{g}} \sim \mathcal{N}N(0, 0.5), \quad g \in [\mathrm{male}, \mathrm{female}, \mathrm{unknown}] \tag{9.11}$$
$$\beta_{\sigma,\mathrm{g}} \sim \mathcal{N}N(0, 0.5), \quad g \in [\mathrm{male}, \mathrm{female}, \mathrm{unknown}] \tag{9.12}$$
$$\beta_{\mu,\mathrm{wd}} \sim \mathcal{N}N(0, 0.5), \quad wd \in \text{weekdays} \tag{9.13}$$
$$\beta_{\sigma,\mathrm{wd}} \sim \mathcal{N}N(0, 0.5), \quad wd \in \text{weekdays} \tag{9.14}$$
$$\tau_\mu \sim \mathcal{C}(0, 1) \tag{9.15}$$
$$\tau_\sigma \sim \mathcal{C}(0, 1) \tag{9.16}$$
$$\gamma_{\mu,\mathrm{w}} \sim \mathcal{N}(0, 1), \quad w \in \text{week nodes} \tag{9.17}$$
$$\gamma_{\sigma,\mathrm{w}} \sim \mathcal{N}(0, 1), \quad w \in \text{week nodes} \tag{9.18}$$
$$\mathcal{S}_\mu = p_i(\text{week nodes}, \tau_\mu * \gamma_{\mu,\mathrm{w}}) \tag{9.19}$$
$$\mathcal{S}_\sigma = p_i(\text{week nodes}, \tau_\sigma * \gamma_{\sigma,\mathrm{w}}) \tag{9.20}$$
$$\mu_i = \mu_{d_i} + \beta_{\mu,a_i} + \beta_{\mu,\mathrm{g}_i} + \beta_{\mu,\mathrm{wd}_i} + \mathcal{S}_\mu(\mathrm{w}_i) \tag{9.21}$$
$$\sigma_i = \sigma_{d_i} + \beta_{\sigma,a_i} + \beta_{\sigma,\mathrm{g}_i} + \beta_{\sigma,\mathrm{wd}_i} + \mathcal{S}_\sigma(\mathrm{w}_i) \tag{9.22}$$
$$\delta_i \sim \mathcal{LN}(\mu_i, \sigma_i) \tag{9.23}$$

In practice, training this model with respect to the huge amount of case data is challenging. One obvious approach is sub-sampling of cases, but due to the complex and time varying distribution of cases in the different districts, we prefer to retain the full data set. Instead, we use several computational techniques to accelerate the MCMC sampling. Due to the large number of parameters, we use a reverse-mode auto differentiation in Julia. To achieve good performance, this requires careful implementation of the statistical model that avoids unnecessary conditionals. Most importantly, we provide a hand-tuned implementation of the log-likelihood of the $\mathcal{LN}$ distribution.

9.2.3. Nowcasting model

In this work, we use a nowcasting model based on GÜNTHER et al. [83], which in turn extends earlier work by HÖHLE and HEIDEN [91]. These approaches use a hierarchical Bayesian model composed of two parts: a model for the delay distribution δ as a function of time ($P(\delta_i = d|x_i = t) := p_{t,d}$) and a model for the expected number of infections per day ($E[N(t,\infty)] =: \lambda_t$).

For the first part, we can either plug in the delay distribution obtained during imputation (cf. Section 9.3.2) or fit a new model, such as the one proposed in GÜNTHER et al. [83], which starts from a discrete time hazard model with $h_{t,d} = P(\delta = d|\delta \geq d, W_{t,d})$ with time- and delay-dependent covariates $W_{t,d}$ and with $\text{logit}(h_{t,d}) = \gamma_d + W'_{t,d}\eta$ for $d = 0, \ldots, D-1$, with bias γ_d and with $h_{t,D} = 1$. In this approach, the covariates $W_{t,d}$ can represent different features of interest. Like GÜNTHER et al. [83], we use a first-order spline to include a general time dependence and factor variables to represent weekdays and holidays. The probabilities $p_{t,d}$ can be obtained from the hazard model through $p_{t,0} = h_{t,0}$ and $p_{t,d} = \left(1 - \sum_{d=0}^{D-1} p_{t,d}\right) \times h_{t,d}$.

For the second part of the hierarchical nowcasting model, we again follow GÜNTHER et al. [83] and use the Gaussian random walk

$$\lambda_0 \sim \mathcal{LN}(0,1) \tag{9.24}$$

$$\lambda_t \sim \mathcal{LN}(\log(\lambda_{t-1}), \sigma^2), \qquad t = 1, \ldots, T \tag{9.25}$$

$$n_{t,d}|\lambda_t, p_{t,d} \sim \mathcal{NB}(\lambda_t \times p_{t,d}, \phi), \qquad t = 1, \ldots, T \tag{9.26}$$

where ϕ is the overdispersion parameter of the negative binomial distribution. Appropriate priors for the model are again taken from GÜNTHER et al. [83] and the implementation referenced therein. We finally obtain $N(t,\infty) = \sum_{d=0}^{D} n_{t,d}$.

The number of parameters of this model grows with $T \times D$, where T is the day at which the nowcast is to be evaluated. Hence, the number of variables grows linearly with each day of the pandemic, rendering sampling increasingly time- and memory-consuming. To prevent this growth from turning nowcasting infeasible, we limit the amount of history we track in the Gaussian random walk, i.e., instead of setting $t = 0$ to the first day of the Covid-19 pandemic, we choose a fixed time interval of size $\Delta > D$ (set to the last $\Delta = 50$ days in our implementation). Since $\lambda_0 = E\left[N(0,\infty)\right]$ equals the number of infections happening at day $t = 0$, we need to adapt Eq. (9.24) accordingly.

9.2.4. Evaluating nowcasts

Evaluating probabilistic predictions requires different scoring functions than point estimates do. For a comprehensive discussion of the topic, we refer the interested

reader to Bracher et al. [27]. Here, we briefly summarize the main evaluation measures used in this work.

Evaluation of now- and forecasts is relatively straightforward if the computational model yields the full predictive probability distribution. Given a test datum y, common measures are the logarithmic score [79]

$$\log\mathrm{S}(y) = \log(p(y)), \tag{9.27}$$

where $p(\cdot)$ denotes the probability density function used for prediction, or the continuous ranked probability score [78]

$$\mathrm{CRPS}(F, y) = \int_{-\infty}^{\infty} \mathrm{d}x \, \{\mathrm{F}(x) - 1_{x \geq y}\}^2, \tag{9.28}$$

where $F(\cdot)$ denotes the cumulative distribution function used for prediction. In Günther et al. [83] the coverage frequency of the 95% prediction interval is used as well.

In many practical situations, the full predictive distribution is not available. In those cases, we typically have access to the predictive mean or median and, optionally, several quantiles of the cumulative distribution function. Assuming that we are given the predictive median m and a collection of K central prediction intervals (PIs) at levels $(1-\alpha_1) < (1-\alpha_2) < \cdots < (1-\alpha_K)$, and weights $w_k, k = 1 \ldots K$, the weighted interval score (WIS) is defined as

$$\mathrm{WIS}_{\alpha_{\{0:K\}}} := \frac{1}{K + \frac{1}{2}} \times \left(w_0 \times |y - m| + \sum_{k=1}^{K} \{ w_k \times \mathrm{IS}_{\alpha_k}(\mathrm{F}, y) \} \right) \tag{9.29}$$

where the interval score IS_α for the central $(1-\alpha) \times 100\%$ PI is defined as

$$\mathrm{IS}_\alpha(\mathrm{F}, y) := (u_F - l_F) + \frac{2}{\alpha} \times (l_F - y) \times 1_{y \leq l_F} + \frac{2}{\alpha} \times (y - u_F) \times 1_{y \geq u_F}$$

and where l_F and u_F denote the $\frac{\alpha}{2}$-quantile and $(1 - \frac{\alpha}{2})$-quantile of F, respectively. Following Bracher et al. [27], we use $w_0 = \frac{1}{2}$ and $w_k = \frac{\alpha_k}{2}$ for $k = 1 \ldots K$, as this choice approximates the CRPS score for large values of K and equidistant α_k.

9.3. Results

The main result of this work is the CorCast system as a computational pipeline. In the following, we will describe its essential components and the application to Germany as a proof of concept.

Our proposed architecture is sketched schematically in Figure A.3. Inference is based on probabilistic programming using Stan [37] and Turing [71]. All system

services are deployed on a Kubernetes cluster [195] for standardized deployment and operations, including simple horizontal scaling on demand. On the cluster, a Pachyderm installation [149] orchestrates compute pipelines and datasets. Pachyderm also keeps track of revisions of individual compute modules and pipelines as well as of input, intermediate, and result datasets. Finally, a Flask- [153] and Dash-based [162] web service, deployed on the Pachyderm cluster as a service, provides a graphical interface to the ingested and processed data as well as to the imputation and nowcast results.

Owing to the high case numbers of the pandemic, each individual module needs to be highly efficient. At the same time, to facilitate extensions and improvements of the pipelines, the module implementations should be intuitively readable and high-level to hide many implementation details required for scaling to modern compute hardware. We achieve this goal through the use of the Julia language [20].

In the following, we will discuss the individual components of this system in more detail.

9.3.1. Data ingest and preprocessing

The first step in our pipeline is the collection of relevant infection data. In Germany, for instance, infection data is collected locally at the health departments at the district level (a political organizational unit below the state). Those departments collect the relevant information and further transmit it to the Robert Koch Institute (RKI), Germany's public health institute. At the RKI, collected information is processed to ensure compliance with privacy regulations, and then published daily. The format of this publication is rather unusual and cumbersome. In addition, the reported dates of interest (infection date, reporting date, and dates of recovery or death) cannot, in general, be uniquely assigned to individual cases[2].

Expanding on our notation from Section 9.1, let us assume that an individual p_i contracts the disease at date x_i and is diagnosed at date d_i. This diagnosis is reported locally, i.e., at district level, at date r_i^l and transmitted to the RKI, which receives the case information at date[3] r_i. Each day, the RKI then publishes a spreadsheet containing information on all previously received cases.

Extracting the required information from these data publications is a challenging task in itself. For more details, c.f. Sec. 9.2.1.

9.3.2. Imputation of disease onset

The goal of this stage is to impute missing disease onset information. Let C denote the set of all cases, $C \supseteq K := \{i \in C | s_i \text{ is known}\}$ the set of cases with known disease onset, and $C \supset U := C \setminus K$ the set of cases with unknown onset. Our task is now

[2] This is probably due to conform with privacy regulations.

[3] Owing to Germany's slightly archaic setup of monitoring of infectious diseases, r_i often differs

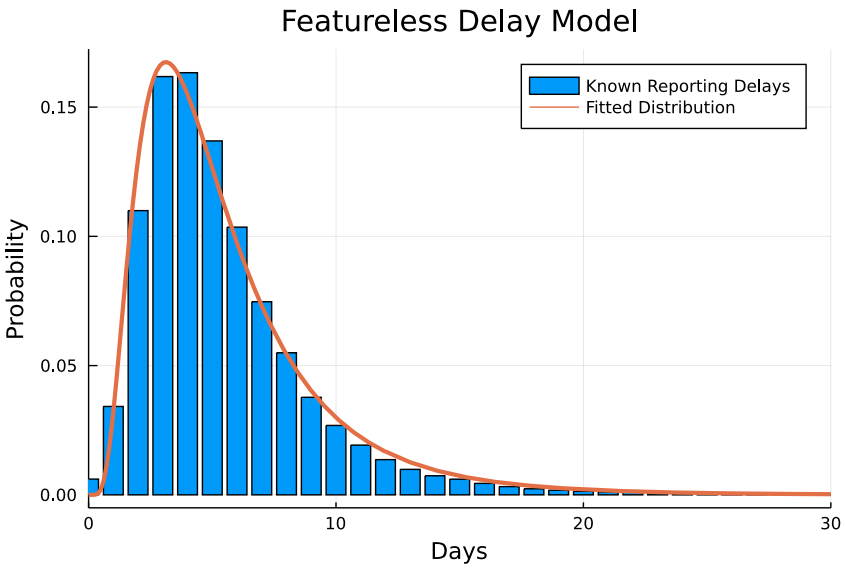


Figure 9.3.: The delay histogram (delays shifted by 1 day to prevent values of zero) against a LogNormal distribution with parameters generated from the mean of 4 MCMC chains with 2,000 samples each, generated by the NUTS Homan and Gelman [93] sampler fed with a subset of 1,918,308 known delays.

to infer $s_i \, \forall i \in U$ based on the information contained in K. To this end, we build Bayesian (hierarchical) models for the *reporting delay* $\delta_i := r_i - s_i$. Inference is then achieved by sampling δ_i from the posterior for each case in U and using δ_i to finally predict[4] $s_i = r_i - \delta_i$.

In our framework, we implement several different statistical models for the delay distribution in two probabilistic programming frameworks (Stan [37] and Turing [71]). The baseline is a trivial model that does not take any covariates of interest into account, see Section 9.2.2 for details. However, even the baseline model fits the data remarkably well. Figure 9.3 shows an example.

In a more complex model we included additional covariates, such as age group, gender, or district of the cases. Since it is reasonable to assume that reporting delays might change over time (e.g., owing to varying test rates or overloads of institutions during high-incidence intervals), the more sophisticated model also includes a slowly varying time-dependent term, e.g., modelled by a spline.

Additionally, the data has an obvious hierarchical component – in Germany,

from r_i^l, with a gap that can vary quite significantly from district to district, weekday to weekday, or week to week.

[4] In practice, we replace δ_i by $\delta_i + 1$ during inference and correct accordingly in the imputation stage. This facilitates inference, as our target distributions typically predict a probability of 0 for delays of 0. These do, however, occur in the data.

districts are located within states. A multi-stage hierarchical model thus seems appropriate to represent dependencies by partial pooling: the means and variances of the delay distribution in each district should ideally be able to learn from the means and variances of each state, and those of the states from those of the country as a whole.

Figure 9.4 shows a graphical representation of the model. See Section 9.2.2 for a detailed description of the model.

We implemented our models in both Stan [37] (using CmdStan.jl [41]) and Turing.jl [71] and draw samples from the posterior using the NUTS [93] sampler to determine probability distributions for the model parameters. From these distributions, we can finally impute infection dates by drawing – for each case with unknown disease onset – delay values from the resulting posterior predictive distribution, conditioned on the appropriate covariates (e.g., the age group, gender, or district associated with the case).

Here it is worth noting that the delay distributions (cf. Figure 9.3) tend to have a rather pronounced peak, typically close to 7 days. If we would sample delay values multiple times for each case with unknown disease onset to average over the posterior predictive distribution, we would thus almost always assign delays of approx. 7 days. Instead, delay values are drawn once per case and used to compute disease onsets. We then group by disease onset and aggregate cases. Repeating this process multiple times allows to compute confidence intervals for the number of new infections per day.

9.3.3. Nowcasting infection numbers

During an epidemic, infection numbers cannot be observed in real time. Due to the reporting delay between the date x_i of disease onset and r_i of receiving the case information at the central case registry (the RKI in Germany), information about current case numbers will necessarily be incomplete. If we assume that the reporting delay δ is bounded by a reasonable maximum D, we can expect, at each day during the epidemic, to be notified about additional cases with disease onset in the last D days (we assume that reporting delays are generally > 1).

We adopted a model from Günther et al. [83], which in turn extends earlier work by Höhle and Heiden [91]. The details are given in section 9.2.3.

To provide nowcasts at the different levels of the hierarchy (district, state, and country), we group the data by district and run NUTS-sampling on each of the resulting 412 datasets (one for each district), followed by one run for each of the 16 states and one for the whole of the country. This process is quite time consuming and takes roughly 5 hours on three AMD EPYC™ 2nd Gen. worker nodes with 8 cores and 32 GiB RAM each.

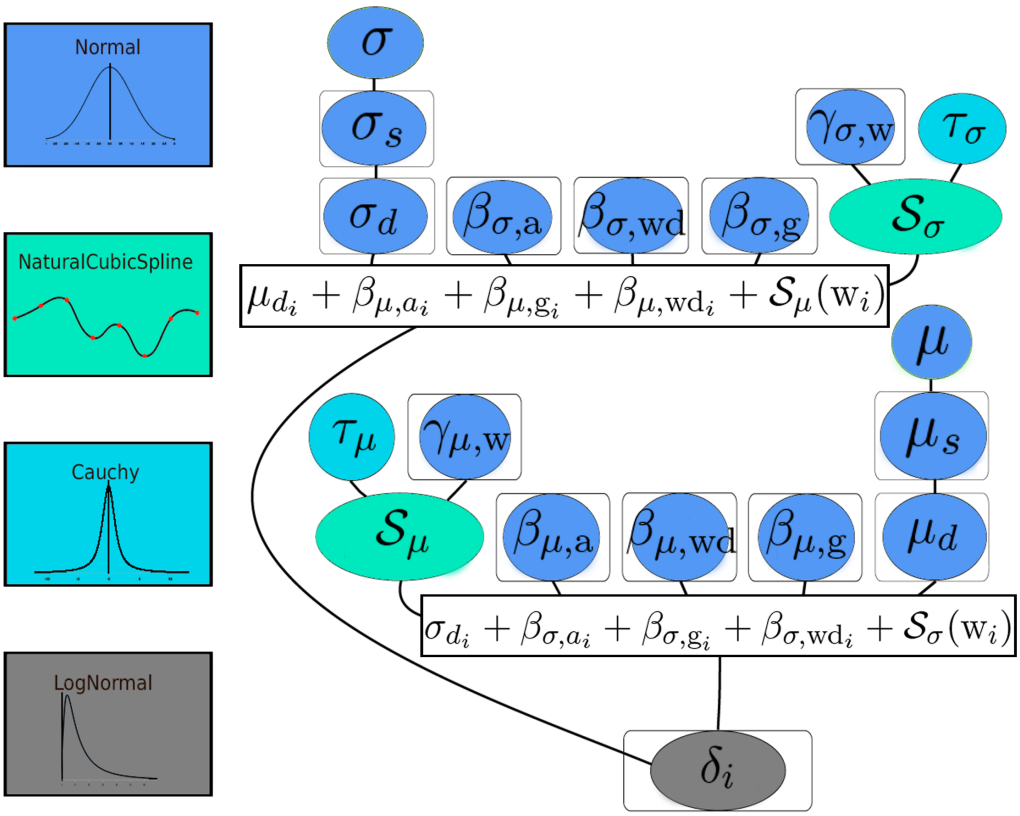


Figure 9.4.: Imputation model with covariates age group, gender, district, weekday, and number of weeks since the start of the pandemic. Ellipses denote random variables, ellipses inside boxes indicate multidimensional variables, the terms in boxes indicate dependent random variables and lines indicate dependencies. The prior distributions used are encoded by color. Note the hierarchical structure of the spatial covariate. Due to computational constraints the model was evaluated in several steps. Details are given in Section 9.2.2.

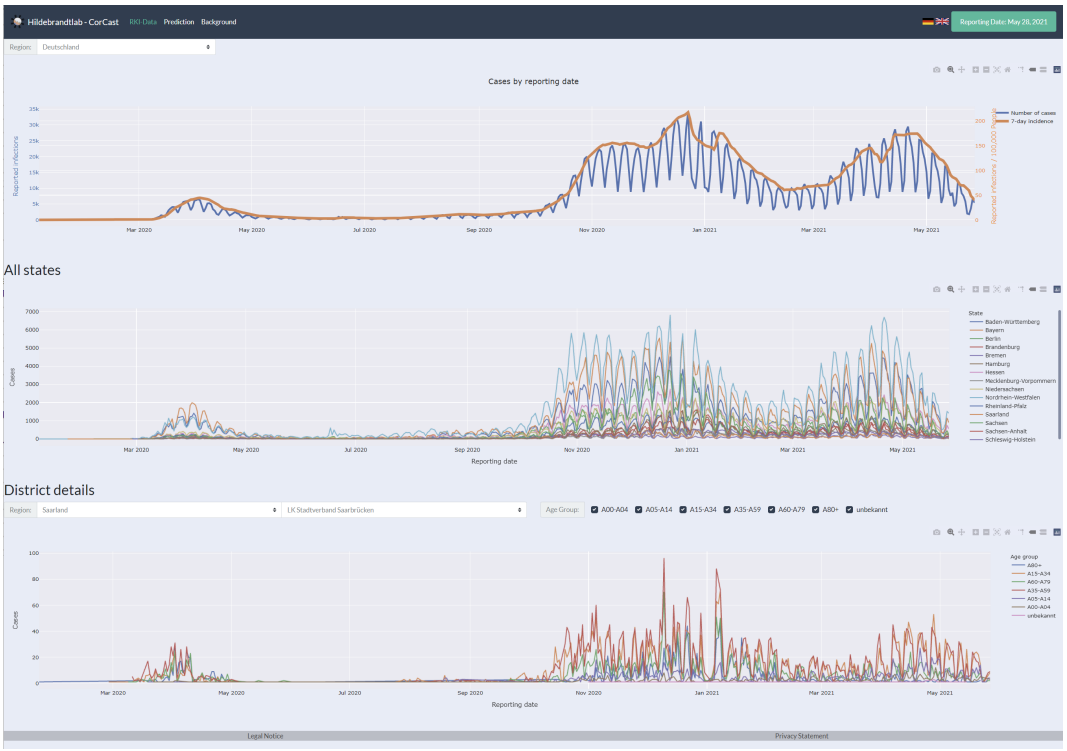


Figure 9.5.: The CorCast dashboard.

9.3.4. Evaluation of nowcast

The evaluation of nowcast results is not a trivial task. In Section 9.2.4, we summarize the relevant literature. CorCast contains an evaluation module that implements the WIS-score described in that section.

9.3.5. Dashboard functionality

CorCast provides a dashboard feature through Pachyderm's service pipeline mechanism. The dashboard is implemented in Flask [153] and Plotly's dash framework [162], and can be easily extended and adapted. The dashboard for the CorCast model of district-level imputation and nowcasting in Germany can be found at https://dashboard.covid19.hildebrandtlab.org/ (cf. Figure 9.5).

9.4. Discussion

Several advantages distinguish the CorCast system described in this work. Here, we want to summarize the most important of these, namely (i) reproducibility (ii)

modularity and extensibility (iii) scalability and (iv) standardized deployment and operations.

9.4.1. Reproducibility

Reproducibility is a cornerstone of science that can be very challenging to achieve. In complex computational systems, it is typically insufficient to merely guarantee accessibility to input data sets and informal descriptions of the employed programs and workflows. Modern scientific computer programs often rely on a large number of third-party dependencies, such as support libraries. Even small variations in the computational infrastructure can lead to differing outcomes, either through the introduction of bugs [21] or through numerical effects that lead, e.g., to convergence to different minima. True reproducibility hence requires reliable versioning of data, programs, and workflows. CorCast achieves this goal by relying on three key technologies: Git, for versioning source code, Docker containers [129] for versioning applications, and Pachyderm [149] for versioning data and workflows.

A side effect of incorporating reproducibility tightly into the system design is a greatly simplified evaluation and comparison of different steps in the nowcasting pipeline. Imagine, for instance, that we want to compare a novel Bayesian model for imputation of disease onset with respect to its influence on the final nowcast. Through the versioning of data and workflows, we can recreate the exact information used to perform the nowcasts at any former date since the initial data commit of the CorCast system, exchange the imputation module through our new candidate, run the full nowcasting pipeline, and finally evaluate the performance and compare it to the historic performance of our former model, which we can again query from the corresponding data repository.

9.4.2. Modularity and Extensibility

CorCast has been designed as a highly modular pipeline, with components responsible for ingesting data, processing and cleaning of the raw inputs, imputation of disease onsets, nowcasting, evaluation, and interactive visualization. Each of these components can be easily replaced, but it is also a simple task to add steps to the pipeline, or to provide alternative implementations for each step in the pipeline (such as competing nowcasting models) and to run them in parallel. To give an example of the implications of this extensibility, we want to stress that the current implementation of CorCast is focused on nowcasting SARS-CoV-2 infection numbers in Germany, but that it would be a simple matter of extending or replacing the ingest nodes of the pipeline to extend it to other countries, or to adapt it for different infectious diseases.

9.4.3. Scalability

While it might not be obvious at first glance, the data generated during a pandemic grows very rapidly. Merely reporting summary statistics, i.e., the number of new infections per day, is typically insufficient, as the progression of the epidemic can vary significantly in different sub-populations (differentiated, e.g., by age, gender, or location). To account for these differences, all differentiating features (including the date of disease onset) will have to be reported. In practice, much of the reporting hence happens at the level of individual cases, which grows rapidly during the pandemic phases of exponential growth. Further complications arise through inconsistencies and retrospective modifications that have to be treated carefully in the data preparation stage. In the case of German infection numbers as provided by the Robert Koch Institute, this requires iterating over all previous case reports, where each case report contains information about every single case since the start of the pandemic, leading to quadratic effort. At the time of writing, the uncompressed data size for this correction step is approximately 43.4 GiB. The large case numbers also render Bayesian inference rather expensive. At the time of writing, imputation of disease onset takes roughly 5 hours, nowcasting roughly 4 hours, on a Kubernetes cluster with one dedicated master and three AMD EPYC™ 2nd Gen. worker nodes with 8 cores and 32GiB RAM each. To handle the large amounts of data and the computational effort, we have taken care to expose opportunities for parallelism, e.g., by separating inference models by geographic region, by running multiple shorter MCMC chains in parallel, or by vectorization- and parallelization primitives inside the computational nodes. Scheduling of parallel computation is achieved through Pachyderm's parallelization primitives. Without these measures, it would be essentially infeasible to produce daily nowcast results at our level of granularity.

9.4.4. Standardized Deployment and Operations

CorCast uses DevOps principles to facilitate deployment and operations. Following an infrastructure-as-code (IaC) approach, all components are provisioned and managed using Terraform [86] and Helm [194] files. All compute nodes are realized as Docker containers and all workflows encoded as Pachyderm pipelines in JSON format. Orchestration and managing of system resources (monitoring, up- and down-scaling, upgrade handling, etc.) are realized through Kubernetes and Prometheus. As a result of the IaC approach, all information required to set up a CorCast cluster is under version control in the CorCast Git repositories. Scaling the existing instance or setting up a new instance on a variety of cloud providers or on premise is trivial. Hence, CorCast can be easily adapted and deployed by researchers interested in epidemics now- or forecasting.

9.4.5. Summary

CorCast is a highly flexible and scalable framework for the implementation of computational statistics and machine learning approaches to epidemiology that provides automatic reproducibility. In addition, the CorCast instance described in this work is, to the best of our knowledge, the only currently available implementation for nowcasting of German Sars-CoV-2 infection numbers that can provide daily updates for disease onset imputation and nowcasts on a district level. In future work, we intend to add additional models to the CorCast system, such as the stable inference of replication numbers, and want to develop pipelines for other geographic regions.

10. Conclusion and Outlook

Summary

In this work, we covered the Bayesian modeling approach from various perspectives.

In chapter 2 we have reviewed the foundations and theoretical advantages of the approach.

Then, in chapter 3 we have reviewed common algorithms for inference, highlighting the different strengths and weaknesses.

Building upon that, in chapter 4, we presented an algorithm for parallel inference and tested it in a case study.

In chapter 5 we reviewed common software packages for probabilistic programming and presented an extension to one of them in chapter 6 that allows for computationally inexpensive online updates.

Next, in chapter 7 we showed an application in surface topography analysis and presented our findings that are action-guiding for future research.

Finally, in chapter 8 we reviewed epidemiological modeling and presented our nowcasting system in chapter 9.

Appendices A and B cover further research that is not related to Bayesian modeling.

Discussion

paraNUTS

Our paraNUTS algorithm achieved considerable speed-ups without too much loss of inference quality. Due to its formulation in terms of the map-reduce paradigm, it is also possible to translate it to other parallel architectures easily.

TuringOnline

Similarly with our TuringOnline package, speed-ups without too much quality loss were achieved, allowing users to update their inference quickly in online setting or to use a hybrid approach.

Surface topography analysis

The work in chapter 7 used advanced Bayesian modeling to leverage prior knowledge for the computation of the magnitude of important, but not directly observable, effects, providing respective uncertainties as well. Thereby, we revealed action-guiding findings for the field of surface topography analysis to ensure reproducible results from future studies.

Corcast

The Corcast system provides the necessary unified treatment of data and models that is extremely important for practical application using a modern cloud-based architecture. Although targeted at the Sars-CoV-2 pandemic, the system is designed

to be adopted to other epidemiological modeling easily.

Appendices While the work in appendix A offers a scalable and flexible approach to signal classification in mass spectrometry raw data, the work in appendix B solves a classification task as well, but here in the field of surface topography analysis using machine learning techniques. It is thus related to the work in chapter 7.

Outlook

paraNUTS In comparison to our paraNUTS algorithm in chapter 4, there are more sophisticated methods described in the literature, see the references in section 4.2. To better assess the different properties it would be beneficial to run a benchmark test against them, which is currently hindered by the fact that not for all algorithms a suitable implementation is available. Nevertheless, we expect paraNUTS to be a relevant contribution in any case since unlike the other approaches it does not require the complex tuning of (potentially) many parameters. Consequently, it should be of greater practical value due to the greater robustness, following the line of the NUTS algorithm which it is based on.

Furthermore, it would be interesting to use several models of different types for the benchmark. For our implementation it would be helpful to allow for an automated split for each function argument.

TuringOnline Our contribution in chapter 6 would benefit from adding a feature that allows for a Kalman filter like approach that runs a certain number of steps online and a full pass run in the background in parallel. Especially interesting would be a combination with the paraNUTS algorithm for a parallelized run on the full pass data.

Surface topography analysis The work in chapter 7 revealed action-guiding findings for the field of surface topography analysis. However, many practitioners in the field are not familiar with Bayesian analysis and thus are sometimes reluctant when confronted with the method. Here communicating the results visually helped, but still there is more work needed to promote the method to a wider audience and to find ways in which one can communicate not in too technical terms but still precise.

Corcast For the Corcast system in chapter 9 it would be highly beneficial to use the developed methods for online updates and parallel processing as over the course of the pandemic the number of cases has risen dramatically such that the standard approaches have become computationally tremendously expensive.

Appendices For the work in appendix A annotated real data is needed to further clarify the abilities of the approach. Similarly, the work in appendix B needs more data to check if more powerful algorithms could be used as well. Another possibility was to reevaluate the data with Bayesian approaches, see e.g. [151] and references therein.

Bayesian analysis workflows As shown in section 2.6, Bayesian analysis follows a typical workflow. Combined with the software tools used to build the Corcast system, see section 9.4, it would be promising to build a pipeline for generic Bayesian analysis workflows. An according

grant proposal has already been submitted.

The Julia programming language offers the possibility to easily run computations on GPUs. Thus it would be interesting to research if additional speed-ups for the paraNUTS and TuringOnline approaches are possible there.

GPU acceleration

Appendices

A. Signal classification in high-throughput mass spectrometry

The results presented here have been previously presented in the following work

> K. Bob, D. Teschner, T. Kemmer, D. Gomez-Zepeda, S. Tenzer, B. Schmidt, and A. Hildebrandt. "Locality-sensitive hashing enables efficient and scalable signal classification in high-throughput mass spectrometry raw data". In: *Submitted to BMC Bioinformatics* (2021)

and will be directly replicated here with minimal changes. K.Bob contributed to the design, implementation, case study and writing of the article.

A.1. Background

A.1.1. Mass spectrometry in proteomics

Valuable information for medicine and design of new drugs for several severe diseases [206] are expected to be gained by new discoveries in proteomics [5][23][161], the field that studies proteins experimentally on a large scale. An experimental technique commonly used in proteomics is mass spectrometry (MS) [2], which allows to separate ionized molecules by their mass-to-charge ratio (m/z) and which can be combined with measurement of other physical and chemical properties.

The overall goal in an untargeted MS-based proteomics experiment is to identify and quantify as many proteins as possible in a given sample with a high quantitative performance in terms of precision and reproducibility. In particular, in bottom-up proteomics proteins are digested into peptides first and then those peptides are measured

If only the mass-to-charge ratio of a large, complex sample of peptides was measured, the resulting signal would be highly convoluted as many peptides have the same or a very similar m/z. In order to minimize overlapping signals and to get further information on the molecules measured, mass spectrometers are coupled with a previous separation device based on orthogonal (ideally) physical and chemical properties. Typically, the peptides are first separated using liquid chromatography (LC), where molecules are separated and gradually eluted at a certain retention time range, e.g., based on their polarity in the commonly used reversed phase (RP) chromatography. More recently, ion mobility (IMS) has become widely accessible in

commercial mass spectrometers as an extra dimension of separation, where ionized molecules are continually separated based on their shape and size before being analyzed in the MS. For instance, in LC-IMS-MS [7][55][128] the retention time and mobility dimensions are recorded in addition to m/z. Figure A.1 shows the experimental workflow in the left column.

Finally, in many experimental setups two types of mass spectra are recorded: so-called MS1 spectra of all the ions, typically for localization of signals of interest denominated precursors and MS2 spectra, where selected (or unselected) precursors are fragmented and the obtained ion patterns (fragmentation spectra) are typically used for identification.

The first step in the analysis of MS1 spectra is to identify regions of interest. These signals form characteristic patterns in the raw data. More precisely, molecules produce so-called isotopic patterns [199] that are caused by the occurrence of different isotopes in the chemical elements. Thus, one expects a pattern of evenly spaced peaks, where the distance between consecutive peaks varies inversely according to the charge state of the molecule (i.e., a spacing of ≈ 1m/z for $z = 1$, ≈ 0.5m/z for $z = 2$, etc.).

The distribution of peak intensity within an isotopic pattern depends on the chemical elements present in a molecule and their respective distribution of isotopes. Although it is possible to deal with the resulting combinatorial complexity [110], often simpler approaches that assume a typical chemical composition are used to model the intensity distribution of the underlying isotope pattern. A common example is the so-called *averagine* model [183].

Due to the coupling with the separation devices, the signals of interest are expected to occur repeatedly over time with the same pattern along the mass axis. Figure A.1 shows an example of a resulting signal in the center and right column.

A.1.2. Problem statement and challenges

The problem to be solved can now be formulated as follows: given MS1 spectra of a high-throughput setup with additional dimensions of separation, classify whether the signals belong to a region of interest or not. This formulation is deliberately more general than a concrete task like deisotoping or charge state detection and should be considered as a preceding step to the other tasks.

On top of the signal processing challenge, another technical problems arises: by introducing additional dimensions of separation (such as LC and IMS), the sizes of single data sets increase. Combined with a growth in the number of data sets measured, the storage used for mass spectrometry data has thus grown tremendously over the past years (cf. Figure A.2) and is expected to grow further. Consequently, dealing with mass spectrometry raw data could strongly benefit from the usage of Big Data technologies, see e.g. [82]. In this approach, data will be stored and processed in a distributed manner, which in turn restricts the algorithms applicable.

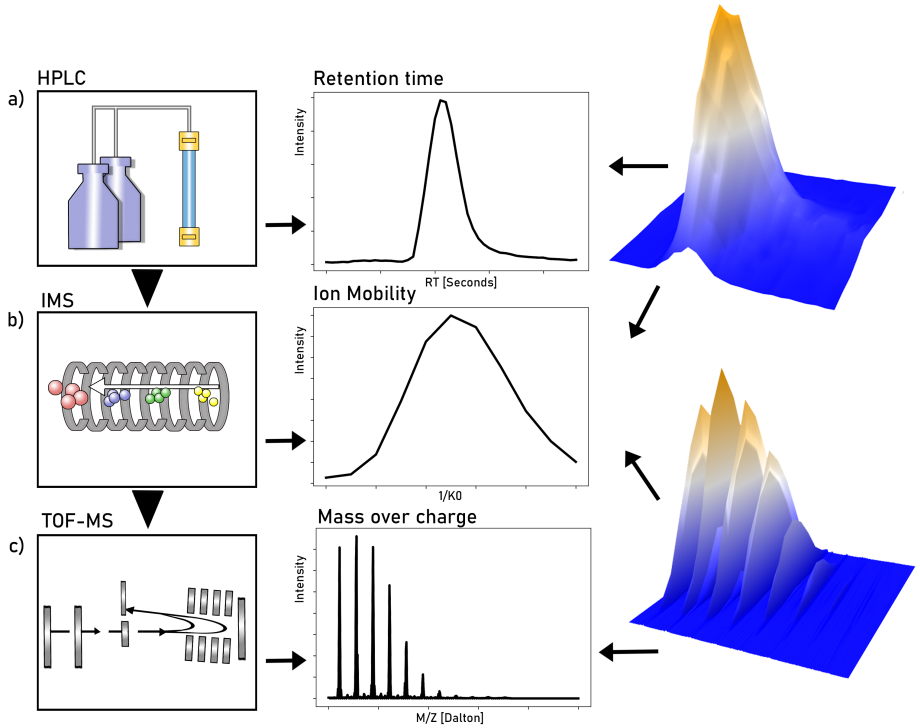


Figure A.1.: Experimental workflow in an LC-IMS-MS setup and resulting signal of interest. **a)** Chemical separation by high-performance liquid chromatography (HPLC) and resulting intensity distribution along retention time. **b)** Subsequent ion-mobility separation (IMS) and resulting intensity distribution along the mobility dimension. **c)** Time-of-flight mass spectrometry (TOF-MS) and resulting isotopic pattern, i.e., evenly spaced peaks with a certain distribution of the peaks envelope. The column on the right shows the signal as a function of two variables, drift time and retention time on the top and mass-over-charge ratio and drift time on the bottom. Note the repeated occurrence of these isotopic patterns in the 3D plot on the bottom right.

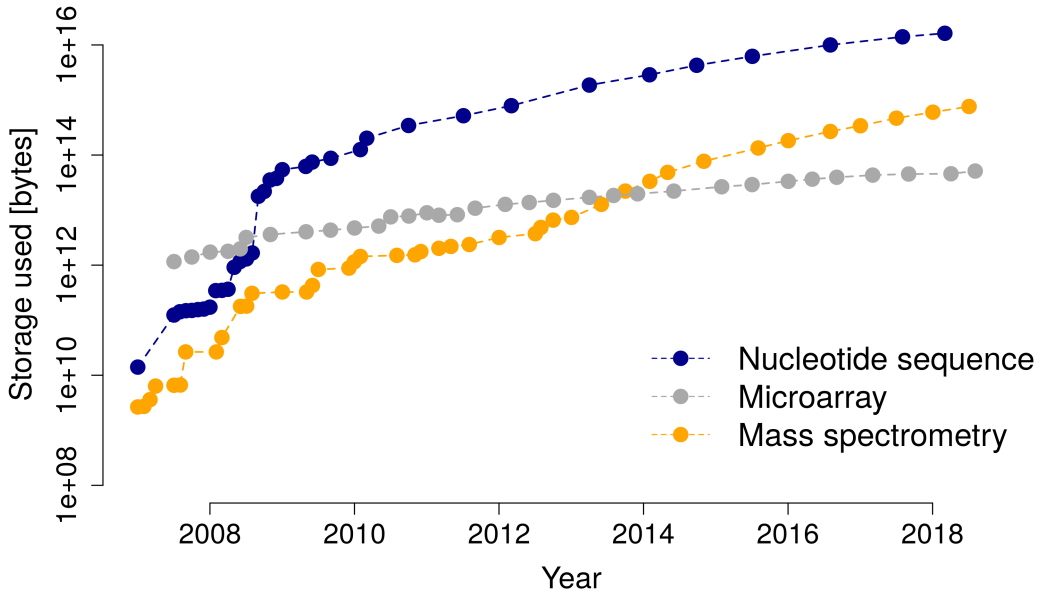


Figure A.2.: Size of recorded data at the European Bioinformatics Institute (EMBL-EBI) over time for different platforms in life sciences. Note the logarithmic scale on the y-axis. Mass spectrometry data has shown an exponential growth. Plot recreated after [43].

A.1.3. Locality-sensitive hashing

An often used Big Data method for the comparison of high-dimensional data is locality-sensitive hashing (LSH) [98][77]. In particular, it is a generic algorithm for finding similar pairs of data points (by some measure) in linear runtime. However, this reduction of runtime comes at the prices that the algorithm is probabilistic in nature.

Owing to its wide applicability, the technique is widely used in different fields, including image retrieval [210], pattern recognition [212], and genome analysis [13][138].

A.1.4. Related work

In the particular context of mass spectrometry, LSH has been used for looking up peptide sequences in databases [60][115], to cluster different spectra for MS1 spectra on LC-MS data [203], and for fast database lookup on MS2 spectra [204].

Previous approaches to signal detection in mass spectrometry raw data either rely on assumptions concerning the isotopic distribution [185] or are based on deep learning [214] and thus lack interpretability.

Concerning the processing of larger data sets, established tools like MaxQuant [45] partially bypass large data sizes by using only parts of the data, i.e., by only looking at every 4th spectrum by default [168].

To the best of our knowledge, signal classification by means of LSH for mass spectrometry raw data has not been treated publicly.

A.2. Results

A.2.1. Approach

Our approach exploits the fact that all signals from a given region of interest are expected to be similar to each other [168], while noise is assumed to be much more random. Thus, classification of the signal is achieved by deciding whether there are similar signals present. Finding similar objects is achieved by locality-sensitive hashing, as it allows to leverage parallel computation.

The overall scheme of the approach, see Figure A.3 for illustration, is the following: a mass spectrometry raw data set is considered to be a set of mass axes for each possible replicated measurement, that is, each retention time for LC-MS data or each retention time and mobility measurement for LC-IMS-MS data, respectively. These mass axes are then cut into small intervals, called windows. For each window several hash values are computed. If two windows have at least one equal hash value resulting from the same hash function they are said to collide. The classification into "true" signal and noise used the following criterion: if a peak lies within a window

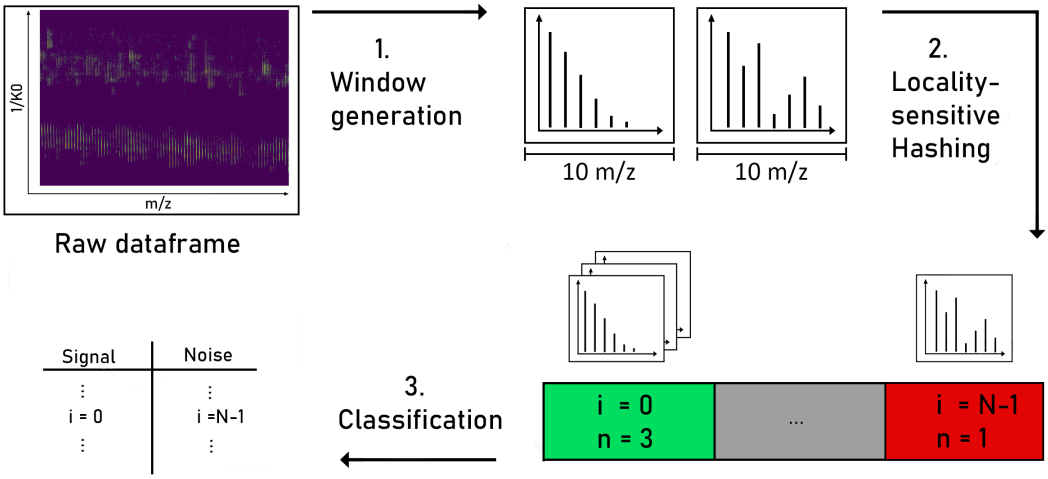


Figure A.3.: Schematic overview of the approach: short intervals (windows) of the several mass axes are considered as smallest building blocks of the data set and generated from a raw dataframe. Then for each window the (several) hash functions h map into the N hash buckets (colored boxes at the bottom right). These are indexed from $i = 0$ to $i = N - 1$ and the number of windows n within a bucket is counted. Finally, if more than one window is mapped in a given hash bucket, all windows inside the box are considered "true" signal.

that collided with any other window it is considered "true" signal, otherwise noise. To facilitate the lookup of collisions, a second map structure is used that maps hash values to their respective number of occurrences.

A detailed explanation of window generation and hashing functions is given in section A.5.

A.2.2. Advantages of the approach

As the hash function of each window can be computed and checked independently of the other windows, the algorithm is embarrassingly parallel in nature. By using an augmented LSH with n AND connectives and m OR connectives, the algorithm allows for tuning of false-positive and false-negative rates.

In particular, our approach assumes no model or distributions for the signal shapes, only similarity. Thus, we are able to distinguish different isotopic patterns as true signal, regardless of the composition, size and charge of the ionized molecule.

[1] Note that the number of windows in the data set influences the collision probabilities as well.

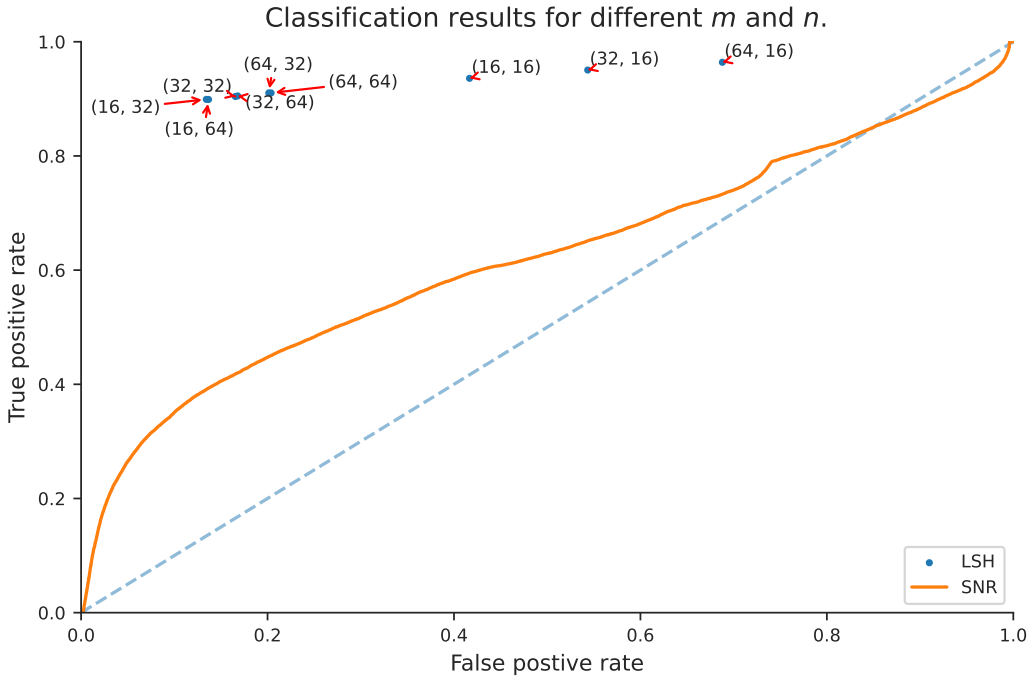


Figure A.4.: Receiver operating characteristic. Single points mark the results of our approach and the tuples denote the number of AND and OR amplifications used. The solid line shows the performance of the intensity-threshold approach and the dashed line the results of random guessing.

A.2.3. Signal classification capability on synthetic data

Figure A.4 shows the receiver operating characteristic (ROC) of a classification task on synthetic data. By varying the amplification parameters m and n of the LSH, different types of discrimination can be achieved, depending on the goal of the user[1].

The achieved performance is in accordance with expected behavior when comparing the implied similarity thresholds in Figure A.5. When setting a low threshold, e.g., $(m, n) = (32, 16)$ (corresponding to the blue line in Figure A.5), a large fraction of windows are considered signal, resulting in very high true-positive and false-positive rates.

A stricter discrimination with $(m, n) = (64, 32)$ improves the performance significantly. This in turn indicates that the typical similarity measure of the data is in the area where the green line in Figure A.5 has a high slope.

Using a very high threshold, e.g., $(m, n) = (32, 32)$ (corresponding to the orange line in Figure A.5), some windows are lost, resulting in a lower true-positive rate. As the false-positive rate shrinks as well, this setting may be employed as a strong filter to reduce large data sets.

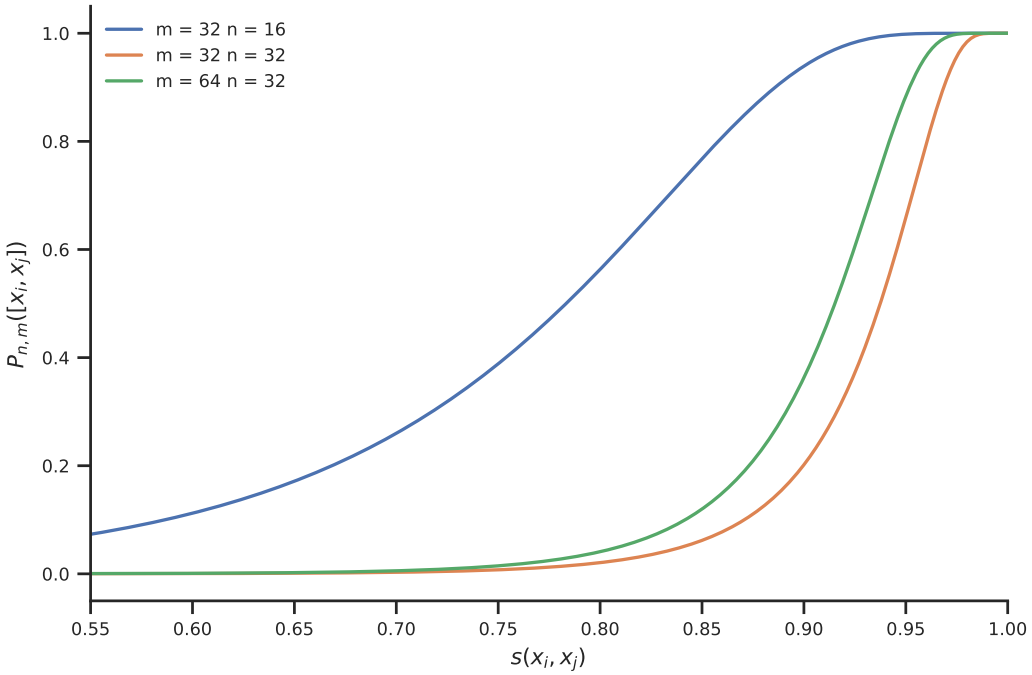


Figure A.5.: Controlling typical similarity of pairs. Probability $P_{m,n}([x_i, x_j])$ to retrieve a pair for several combined hashes as a function of similarity $s(x_i, x_j)$. By appropriate choice of m and n, found pairs have a high probability of having at least a certain similarity. Note that the x-axis starts at $s(x_i, x_j) = 0.55$, for values smaller than that all curves are almost zero.

The performance of our approach was compared to the common approach of filtering signal by an intensity threshold. While the overall behavior of its ROC curve is similar to the one of our approach, the performance is considerably better.

Several synthetic data sets with different noise characteristics were created and all main findings persisted.

A.2.4. Filtering capability on real data

Next, we conducted a study to determine the capability of our approach on real data. An option would have been to use a file with peptides identified by any protein identification software, such as MaxQuant. Nevertheless, this would introduce a bias towards high quality ions and only those present in the protein database would be considered. Therefore, we decided to employ the precursors fragmented during the DDA-PASEF analysis, which are selected based on the ion charge, intensity, and ion mobility profile. We computed the rate of windows with selected precursor ions kept by our filtering and the reduction in data set size after the filtering, as shown in Figure A.6. As can be observed, reduction rate has to be balanced with not loosing too much relevant signal by choosing suitable values of m and n. That way it is possible to reduce the amount of data points in a frame by at least a half, without loss of almost any vendor selected peaks. We found both the reduction rate and the rate of selected precursor ions kept to be between approximately 70% and almost 100%, where the two are inversely related, i.e. higher reduction rate implies lower keeping rate and vice versa.

For details on the evaluation see section A.5.

A.2.5. Using averagine modeled reference patterns for deisotoping

Translating slices of mass spectra into a set of representative keys is not limited to self-similarity detection. Other approaches used LSH for fast candidate identification of fragment-spectra, see for example [203] [204]. There, first a look-up database is built by mapping all reference spectra into LSH-key representations. By using LSH, unidentified candidate spectra only need to be matched with a subset of candidates in the database, greatly reducing the total number of comparisons. For a more detailed explanation of algorithmic solutions to spectral identification in general, the interested reader is referred to [122].

We built on this idea of a pre-computed reference, but used typical isotope patterns to find peptide ions of a specific charge state in precursor spectra in contrast to fragment spectra. These synthetic isotope patterns are hashed to form a reference database, which then can be used to detect isotope patterns by collision of hashes in sets of spectrum windows. By design, the approach is more restrictive then our first presented strategy of self-collision. This is due to the fact that it requires a window to be similar to one of the modeled peak distributions.

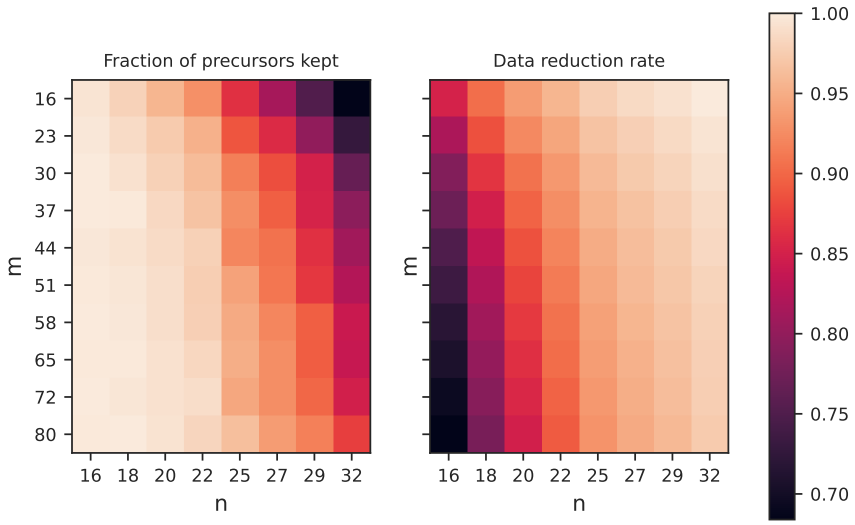


Figure A.6.: Evalutation on real data: fraction of precursors kept (left panel) and data reduction rate (right panel) as functions of m and n. For certain choices of m and n only few windows containing selected precursors are lost, while the data reduction rate is very high, emphasizing the intended usage of our approach as a prefilter. However, m and n can also be varied to balance the trade off between recovery rate and reduction.

We used a subset of the real data set mentioned before. The vendor software selected $N = 6902$ peptides for fragmentation during DDA-PASEF in that subset. We were able to compare those with isotopic patterns found by our approach. When m and n are set accordingly, $k_1 = 6200$ of those $N = 6902$ peptides were considered matched by our initial metric ($\approx 91\%$ of N). $k_2 = 5125$ of the matches k_1 had the correct charge state ($\approx 74\%$ of N). $k_3 = 3419$ of the matches k_2 perfectly matched the monoisotopic m/z and charge state ($\approx 50\%$ of N).

For details on the evaluation see section A.5.

A.2.6. Scalability and acceleration

Figure A.7 shows the scalability of the approach for different choices of m and n on parts of a measured data set. In this double-logarithmic plot, the wall-clock time of the implementation is at first a decreasing function of the number of threads used and then saturates. This shows that the implementation scales out well up to a certain point, at which the parallelization overhead cancels the gain in speed. For details regarding the setup, see section A.5.

Furthermore, our implementation supports acceleration by graphics processing units (GPUs) by building on the tensorflow 2 software library [1].

A.2.7. Software design and availability

We built on top of the low-level C++ API of OpenTIMS [111] for data access. Furthermore, we used pybind11 [100] to interface C++ with the Python programming language and provide both the C++ and the Python code as a Python package. This should make it easy to integrate our tool into the well established, Python-centric data science stack.

A.3. Discussion

While the evaluation on both synthetic and real data showed that locality-sensitive hashing could be used in a promising way to detect signals in mass spectrometry raw data, two challenges are expected when applying the method to real world data.

First, in some mass spectrometry setups so-called chemical noise is present. These are signals that originate in chemical impurities in the measurement workflow and are characterized by repeated occurrences. Thus, the chemical noise is self-similar and would be classified as a “true” signal accordingly by this approach. Since the removal of contaminants can also be performed in subsequent step in a processing pipeline, this is of lesser concern, but should be kept in mind. Second, signals with a low relative intensity to noise peaks could be lost, as the similarity measure on which the LSH is based drops.

While enabling the evaluation of the proposed approach in a controlled environment,

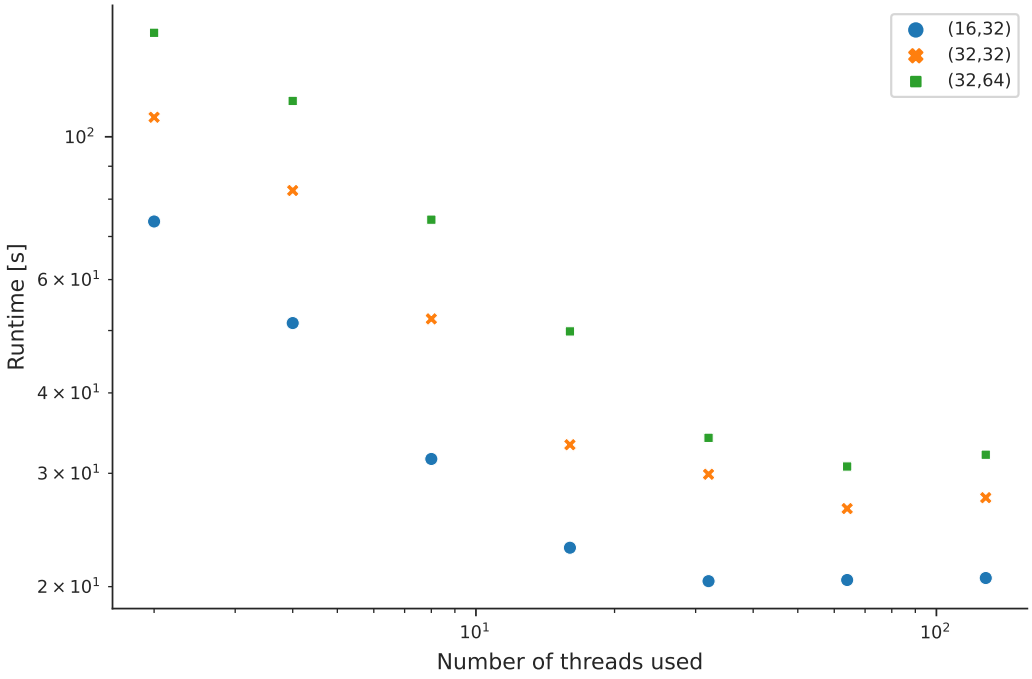


Figure A.7.: Scalability for different m and n: wall-clock time of the implementation in seconds as a function of the number of threads used. More precisely, the cumulative runtime for processing of 100 randomly selected precursor frames is shown. Note the logarithmic scales on both axes. First the wall-clock time is a decreasing function of the number of threads used and then saturates. This shows that the implementation scales out well up to a certain point, at which the parallelization overhead cancels the gain in speed. See the text for the setup used.

synthetic data cannot fully reflect the behavior of real data. However, as we could not find a gold-standard ground-truth peak-level annotation of data, the evaluation on real data has it's own challenges as well. Thus we used the precursor ions selected by the vendor software as a reference to measure the performance of our approach.

Using vendor selected peaks from a DDA experiment comes with its own challenges. First, the annotated masses correspond only to peptides selected for fragmentation and second, those peptides likely stem from high intensity signals. Nonetheless, being able to retrieve those signal with our approach that is not explicitly based on absolute intensity values, increases confidence in its general applicability.

Additionally, we indicate a potential application for desiotoping and charge state detection. This was demonstrated by our prototypic implementation of a reference based pattern matching using isotopic patterns following the averagine model. The reference search is not yet perfect, for instance we observed that for some cases the the monoisotopic m/z value identified is shifted by multiples of m_n/z. Furthermore, charge state identification sometimes fails as well.

Lastly, our implementation is generic enough to be used not only for signal filtering but could also be further extended for additional tasks.

A.4. Conclusions

Due to the rapidly growing amount of mass spectrometry data, the analysis of mass spectrometry raw data could greatly benefit from Big Data methods, most notably implying distributed data storage and highly scalable algorithms.

In this study we showed that locality-sensitive hashing is a desirable approach for signal classification in mass spectrometry raw data. It allows for scalability and provides an approach to signal classification that has a strong focus on self-similarity rather than model assumptions as an intrinsic property of the data.

We propose an implementation using a Big Data framework, such as Apache Spark [211], to facilitate testing on many large data sets from different types of mass spectrometry measurements.

A.5. Methods

A.5.1. Mass axis windows

For the types of mass spectrometry data considered here, the setup of the mass spectrometer gives rise to a hierarchical structure of the data. In the case of LC-IMS-MS, data is continuously acquired as individual scans across the three dimensions. For each retention time several mobility bins are recorded and in turn for each mobility bin the full spectrum along the mass axis is acquired. In order to allow detection of smaller regions of interest, the algorithm works on short compact subsets (intervals)

of the mass axis, henceforth referred to as *windows*. One such window, represented by a list of tuples $(\,\mathrm{m/z}, i)$, is the single datum considered by the algorithm for similarity search.

Window generation

The set of windows is created by dividing all recorded mass axes into windows. In order to avoid missing an isotopic pattern by distributing it into two windows, a second set of windows that is offset by half a window length is created, such that the mass axes are covered in an overlapping fashion.

The length of the windows should be wide enough to capture a whole isotopic pattern, if the windows are applied in an overlapping fashion and small enough such that the chance of having several isotopic patterns in a window is small. For an assumed pattern length of up to 5Da a window length of 10Da is considered useful.

Binning

The calculation of a window's hash value requires the mass axis in a binned form with equally spaced bins. Finding an optimal binning scheme is by no means trivial: a very fine binning resolution makes the algorithm less robust and increases the computational cost, while a very coarse binning resolution yields an increased loss of information. A binning resolution of 0.1m/z was considered suitable as, on the one hand, it still resolves isotopic pattern with up to charge state five (corresponding to a spacing of 0.2m/z) and, on the other hand, keeps the computational load feasible.

A.5.2. Locality-sensitive hashing

Used similarity measure

Two windows W_i and W_j shall be considered similar when their mass spectra have the same shape but not necessarily the same overall scale. Therefore, the similarity function $s(W_i, W_j)$ used is the cosine similarity

$$s(W_i, W_j) = \frac{\langle \mathbf{I}_i, \mathbf{I}_j \rangle}{\|\mathbf{I}_i\| \|\mathbf{I}_j\|}, \tag{A.1}$$

where $\mathbf{I}$ denotes the intensity array of a binned window, $\langle \cdot, \cdot \rangle$ the standard scalar product and $\| \cdot \|$ the Euclidean norm. As a direct consequence from the linearity of the scalar product and norm, the cosine similarity is scale invariant:

$$s(\alpha W_i, \beta W_j) = \frac{\langle \alpha \mathbf{I}_i, \beta \mathbf{I}_j \rangle}{\|\alpha \mathbf{I}_i\| \|\beta \mathbf{I}_j\|} = \frac{\langle \mathbf{I}_i, \mathbf{I}_j \rangle}{\|\mathbf{I}_i\| \|\mathbf{I}_j\|} = s(W_i, W_j), \tag{A.2}$$

for $\alpha, \beta > 0$. Thus, the findings of the algorithm are independent of the absolute intensity values.

Hash function and amplification

The appropriate family of hash functions for the cosine similarity is the random projection hashing [40]. The hash function is given by $h(\cdot) = \text{sign}(\langle \cdot, r \rangle)$ with the components of the vector r being random samples from a standard normal distribution.

To control for false positives or false negatives, an augmented LSH with n AND connectives and m OR connectives is used. This means that instead of computing a single hash function $m \times n$ hash functions are calculated and divided into m arrays of length n. Two objects x_i and x_j are now considered collided if all n hash functions of a block yield the same value. By choosing m and n appropriately, a sharp sigmoid shape of the collision probability $P_{m,n}([x_i, x_j])$ can be achieved, which effectively translates into a similarity threshold. Figure A.5 shows $P_{m,n}([x_i, x_j])$ for different values of m and n.

A.5.3. Intensity thresholding

A very simple and general approach to signal classification is by means of thresholding with respect to a signal-to-noise ratio [10]. If one assumes a global noise estimate, this reduces to a thresholding with respect to the intensity of each peak.

A.5.4. Data generation

The synthetic data sets consist of windows of length of 10m/z, with a binning resolution of 0.1m/z.

In each set, two types of data were created: firstly, windows containing no isotopic patterns and noise only. Secondly, windows containing both isotopic patterns and noise. The label of each peak ("signal", if it is part of an isotopic pattern, "noise_2", if it is a noise peak in a window that contains an isotopic pattern, and "noise_1", if it is a noise peak inside a window without an isotopic pattern being present) was stored for later evaluation. In total 18445 windows per data set were created, of which 14107 contained noise only and 4338 contained both signal and noise.

In the different data sets the intensities of the "true" signals were scaled differently, such that the maximum signal peak has an intensity of 1000, 500, 250, 125, 64, or 32, respectively.

Modeling noise

The noise signal is assumed to consist of independently sampled peaks that share the following properties: the number of peaks per window, k, is given as

$$k \sim \text{Pois}(\lambda_p) + 1,$$

where Pois denotes a Poisson distribution with mean λ_p. Our data was generated using $\lambda_p = 4$.

The location of each peak was sampled uniformly within the window and the intensity value i of each peak was sampled from

$$i \sim \mathrm{Exp}(\lambda_e),$$

where Exp denotes a exponential distribution with mean λ_e^{-1}. Our data was generated using $\lambda_e = 15$.

Modeling isotopic patterns

For the "true" signal the averagine model [183] was used. The monoisotopic peak was placed in the middle of the window and the contribution of the next five peaks was considered. In order to model the repeated occurrences of patterns, two copies with all peaks scaled to half intensities were added. Finally, a noise signal, individually sampled according to description above, was added to every true pattern window.

The monoisotopic masses were ranged from 150u to 5000u in steps of 10u. Charge states from $1e$ up to $5e$ were included if the resulting m/z was in the interval $[150\mathrm{m/z}, 2000\mathrm{m/z}]$.

A.5.5. Classification by collision

In order to classify windows into "noise" or "signal", for each window in the data set (several) hash values are computed. Then the number of occurrences of each hash value is counted and all hash values that occurred more then one time are stored in a table, the so-called collision table. A window in question is called "signal" if at least one of its hash values can be found in the collision table, otherwise it is called "noise".

A single peak in the data set is classified by whether there was a collided window that contains the peak. This is especially important as peaks can be part of several windows due to the overlapping window approach.

A.5.6. Evaluation on synthetic data

Although the synthetic data provides ground truth labels, a meaningful computation of true-positive and false-negative rates is not straightforward.

Since in our approach all peaks in a window will be assigned the same label, the following problem is caused: when signal and noise is present in a window, peaks with label "noise_2" will be considered "true" as well, which would result in a high false classification rate. For work on real world data however, the inability to remove noise peaks among isotope patterns is typically cured by the fact that subsequent processing steps like feature finding can take the multidimensional signal shape into account, which facilitates the removal of remaining noise peaks.

Thus, in order to learn about the false classification rate of actual interest, peaks with label "noise_2" were not considered for the computation of classification rates both for the intensity thresholding and our approach.

The presence of peaks with label "noise_2" is nevertheless important to test whether whole patterns are missed due to noise in the same window.

A.5.7. Real data set used

As test data a single MS1 frame of a nanoLC-TIMS-MS/MS (DDA-PASEF [128]) analysis of HeLa whole proteome digest was used and raw data access was enabled by OpenTIMS [111]. HeLa cells were lysed in a urea-based lysis buffer (7 M urea, 2 M thiourea, 5 mM dithiothreitol (DTT), 2% (w/v) CHAPS) assisted by sonication for 15 min at 4 °C in high potency using a Bioruptor instrument (Diagenode). Proteins were digested with Trypsin using a filter-aided sample preparation (FASP) [209] as previously detailed [56]. 200 ng of peptide digest were analyzed using a nanoElute UPLC coupled to a TimsTOF PRO MS (Bruker). Peptides injected directly in an Aurora 25 cm x 75 µm ID,1.6 µm C18 column (Ionopticks) and separated using a 120 min. gradient method at 400 nL/min. Phase A consisted on water with 0.1% formic acid and phase B on acetonitrile with 0.1% formic acid. Sample was injected at 2% B, lineally increasing to 20% B at 90 min., 35% B at 105 min., 95% at 115 min. and hold at 95% until 120 min. before re-equilibrating the column at 2%B. The MS was operated in DDA-PASEF mode [128], scanning from 100 to 1700 m/z at the MS dimension and 0.60 to 1.60 1/k0 at the IMS dimension with a 100 ms TIMS ramp. Each 1.17 sec MS cycle comprised one MS1 and 10 MS2 PASEF ramps (frames). The source was operated at 1600 V, with dry gas at 3 L/min and 200 °C, without nanoBooster gas. The instrument was operated using Compass Hystar version 5.1 and timsControl version 1.1.15 (Bruker). All reagents and solvents used were MS-grade.

A.5.8. Evaluation on real data

The vendor software returns a list of peptides selected for fragmentation. For these peptides the m/z coordinate of the monoisotopic peak and apex of ion mobility distribution are given on a frame basis. In the following, we consider the peptides as 2d data points with the corresponding coordinates.

For our approach we built the mass axis windows as described above with a window length of 10m/z, but only kept windows, for which there was at least one peak exceeding an intensity of 24 and more than 2 peaks in total. Our signal classification returns the mass axis bin and scan of the windows considered signal. We used the center of the window and the scan number as coordinates to represent the found windows as 2d data points as well.

A peptide is considered matched, if it's nearest neighboring window is closer than

5 units in Manhattan distance. Finally, we calculated the reduction rate, i.e. one minus the fraction of windows considered signal of all windows in a frame. Also we computed the fraction of peptides considered matched.

We averaged the results on 50 randomly selected frames, excluding the first 1500 frames from the data set. The whole procedure was repeated for varying m and n.

A.5.9. Isotope pattern reference construction and evaluation of reference search

The averagine model as proposed by [183] was used to create all reference patterns. It was sampled with a step size of 10^{-5} in a range from -1 to $+9$ Da around the monoisotopic peak and afterwards binned to a resolution of 10^{-2} Da. This was done for all masses from 150 to 1800 Da and for all charge states from 1 to 5.

We then applied our hashing strategy to those synthetically generated mass spectra the same way it was applied to self-similarity detection. When an unidentified window is now hashed, its key-set can be used to select a set of candidate isotope-patterns, for which cosine similarity is calculated explicitly. Should a certain threshold of similarity be surpassed (here, we used 0.6), the highest scoring reference pattern is returned together with its charge state and monoisotopic mass.

Again we used the list of peptides selected for fragmentation by the vendor software, with the m/z coordinate of the monoisotopic peak, the apex coordinate of the ion mobility distribution and now additionally the charge state. In the following, we considered the peptides as 3d data points with the corresponding coordinates. A peptide selected by the vendor software is considered matched, if the Manhatten distance to the nearest neighbor in that particular space is less then a threshold, here 10 units.

A.5.10. Scalability study

The scalability study was performed on a single Ubuntu 20.04 LTS machine with AMD Ryzen Threadripper 3990X processor featuring 64 physical cores @2.9 GHz. Overall, enabled hyper-threading allows for the parallel execution of 128 threads. The machine is further equipped with 256 GiB (8x 32 GiB) of DIMM DDR4 @2667 MHz main memory.

As test data 100 randomly chosen MS1 frames of the real data set was used.

B. Classification of contact material from surface parameters

This chapter deals with research that is closely related to that in chapter 7 with respect to the subject of the study, but differs in the analysis approaches.

The results presented here have been previously published in the following work:

> A. Pedergnana, I. Calandra, A. A. Evans, K. Bob, A. Hildebrandt, and A. Ollé. "Polish is quantitatively different on quartzite flakes used on different worked materials". In: *PLOS ONE* 15.12 (Dec. 2020). Ed. by M. Peresani, e0243295. issn: 1932-6203. doi: 10.1371/journal.pone.0243295

K.Bob designed and implemented the machine learning analysis and contributed to the writing of the article.

B.1. Machine learning approach

As mentioned in section 7.1, one goal of surface topography analysis is to infer contact materials by their traces on the object of interest. With the availability of surface parameters like those in Table 7.1 as a quantitative description of a given surface, classification of contact material through machine learning suggests itself. Following the common naming convention, from now on we will refer to the type of contact material as labels and the values of the surface parameters recorded as features. By performing experiments with known contact labels, data with both known features and labels was generated, resulting in a supervised learning setting for classification.

B.2. Data generation

Experiments

In those experiments, five different materials (wood, bone, antler, fresh and dry skin and cane) were worked for an hour with several quartzite flakes from two varieties (VSH4 and A35). One control sample per variety were left unused. Afterwards, all samples were cleaned according to a protocol, see the corresponding section in [158] for details.

Preprocessing

In order to obtain the required surface parameters, the surfaces were scanned with a laser scanning confocal microscope and some sub-areas were extracted. Next, the software ConfoMap (version 7.4.8633) was used to calculate surface parameters. As

there were some potentially problematic sub-areas, the data set was split into a "full dataset" with $n_f = 78$ data points and "restricted dataset" with $n_r = 70$ data points. Unfortunately not all surface parameters could be measured reliably on all sub-areas. The missing points could be meaningfully filled for some surface parameters (Asfc and Smfc on the full dataset and HAsfc9 on both datasets), other parameters (HAsfc81, Periodicity, Period and Direction.of.period for both datasets) had to be removed from downstream analysis. See again the corresponding sections in [158] for details. Finally, $k = 33$ surface parameters and the quartzite type could be used as features on both datasets.

B.3. Analysis

Challenge Due to to the costly experimentation and preprocessing, only a small number of data points – both in general and per class – but with large number of parameters were available. This resulted in a unfavorable ratio of data points to model parameters.

B.3.1. Feature selection

Consequently, we approached the problem by selecting only some parameters, both to prevent overfitting and for practical applications where one does not want to measure all parameters. Thus, the amount of information on the labels that can be gained by each feature was measured and features were selected by their predictive power.

Mutual information In order to achieve this, we used the mutual information[1], which is defined as follows: given two continuous random variables X and Y with joint probability density $\pi_{(X,Y)}$ and marginal densities $\pi_X(x)$ and $\pi_Y(y)$ the quantity

$$\mathrm{I}(X,Y) = \int_{\Omega_Y} \int_{\Omega_X} \pi_{(X,Y)}(x,y) \log_a \left(\frac{\pi_{(X,Y)}(x,y)}{\pi_X(x)\,\pi_Y(y)} \right) \mathrm{d}\mu_X(x)\,\mathrm{d}\mu_Y(y) \tag{B.1}$$

is called the mutual information [107]. Using the base $a = 2$ results in the unit "bits", a commonly used unit for information. For independent variables, the mutual information is zero, while for dependent variables higher values indicate a larger amount of shared information. Due to the structural similarity to the Shannon entropy

$$\mathrm{H}(X) = -\int_{\Omega_X} \pi_X(x) \log_a \left(\pi_X(x)\right)\, \mathrm{d}\mu_X(x) \tag{B.2}$$

the mutual information can be expressed in terms of entropies [107] as

$$\mathrm{I}(X,Y) = \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y), \tag{B.3}$$

[1] For other approaches to feature selection see e.g. the work by Prado [166, Ch. 6]

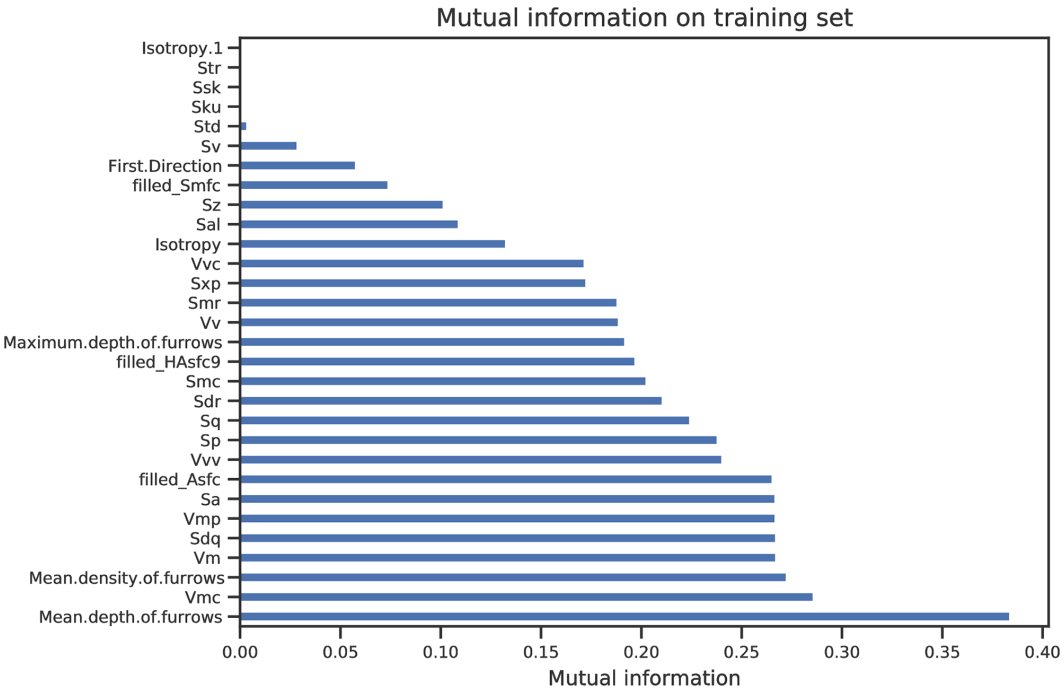


Figure B.1.: Mutual information between features and labels on the training set without the quartzite type as a feature. Image taken from [158, Fig. 2]. See text in subsection B.3.1 fur further information.

where

$$\mathrm{H}(X,Y) = \int_{\Omega_Y} \int_{\Omega_X} \pi_{(X,Y)}(x,y) \log_a \left(\pi_{(X,Y)}(x,y) \right) \, \mathrm{d}\mu_X(x) \, \mathrm{d}\mu_Y(y). \tag{B.4}$$

Shannon [184] shows in sections 12 and 24 of his landmark publication that the same expression as in equation (B.1) describes the rate of actual transmission of information when a source emits realizations of X over a noisy communication channel such that realizations of Y are received instead. This image provides a nice interpretation of the mutual information.

Compute

For computing the mutual information on the data, we used the implementation in the *scikit-learn* package [159] that is based on the works [107] and [173]. The full source code related to our publication is available at https://doi.org/10.5281/zenodo.4249219. The resulting estimates are shown in Figure B.1.

Selection

Those surface parameters that showed a considerably higher value in Figure B.1 were chosen as features.

In order to evaluate the selection approach, the training was performed with all features as well for comparison.

B.3.2. Classification task

Having selected the most promising features, two common classification algorithms were used: decision tree and support vector machine, which will be briefly introduced now. For the following explanations, we will use the notation $\boldsymbol{x} = \{x_i\}_{i=1}^n$ for the n features and y for the label of a single given datum.

Decision tree

The decision tree algorithm [169] [87, chapter 9] is a simple yet powerful method that can be used both for regression and classification tasks. For the latter, we used a version that works as follows:

Concept — The data set gets repeatedly split into two disjoint sets. Calling the data set under current consideration $\mathcal{S}_i$, it is split into the sets $\mathcal{L}_i$ and $\mathcal{R}_i$ by choosing thresholds t_i on a single feature x_k in $\boldsymbol{x} = \{x_j\}_{j=1}^n$ and assigning all data points in $\mathcal{S}_i$ to $\mathcal{L}_i$ if $x_k \leq t_i$ and to $\mathcal{R}_i$ otherwise. The splitting is repeated until only a single data point is left in $\mathcal{S}_i$, or alternatively for a predefined number of steps.

Representation — When interpreting the sets as nodes and linking the $\mathcal{S}_i$ with the respective $\mathcal{L}_i$ and $\mathcal{R}_i$, the resulting graph is a tree with the full data set as the root, giving rise to the name of the method. The learned labels are assigned to the leaf nodes and a given input is traced through the tree until it reaches a leaf node. Figure B.2 shows an example.

Computation — Selecting the right thresholds is achieved by optimizing a scoring function. In this case the entropy $\mathrm{H}(P_Y(y))$ from equation (B.2) was minimized in each node with estimated probabilities

$$P_Y(y) = \frac{1}{n} \sum_{i=1}^{n} \chi_{\{y\}}(\xi_i), \tag{B.5}$$

where ξ_i denote the labels of data points in a node.

For the detailed algorithm see again [169] and [87, chapter 9], for the actual computation we used the implementation in the *scikit-learn* package [159].

Discussion — An advantage of the decision tree is that due to the graphical representation, see again Figure B.2, the approach is easier to communicate to a broader audience. In contrast, other more advanced classifiers are often less accepted as they are considered black-boxes and thus less trusted. For a general review on the topic of interpretable machine learning see e.g. the work of Murdoch et al. [139] and references therein. Furthermore, its simple structure leads to only one hyperparameter, namely the depth of the tree, making the tuning easier.

A disadvantage of the concept is that more complicated decision boundaries in the feature space can only be approximated by deep trees which are more prone to overfitting. Although this can be cured by using a an ensemble of trees, called random forest, the resulting classifier is in turn too complicated to be easily understandable by the general audience.

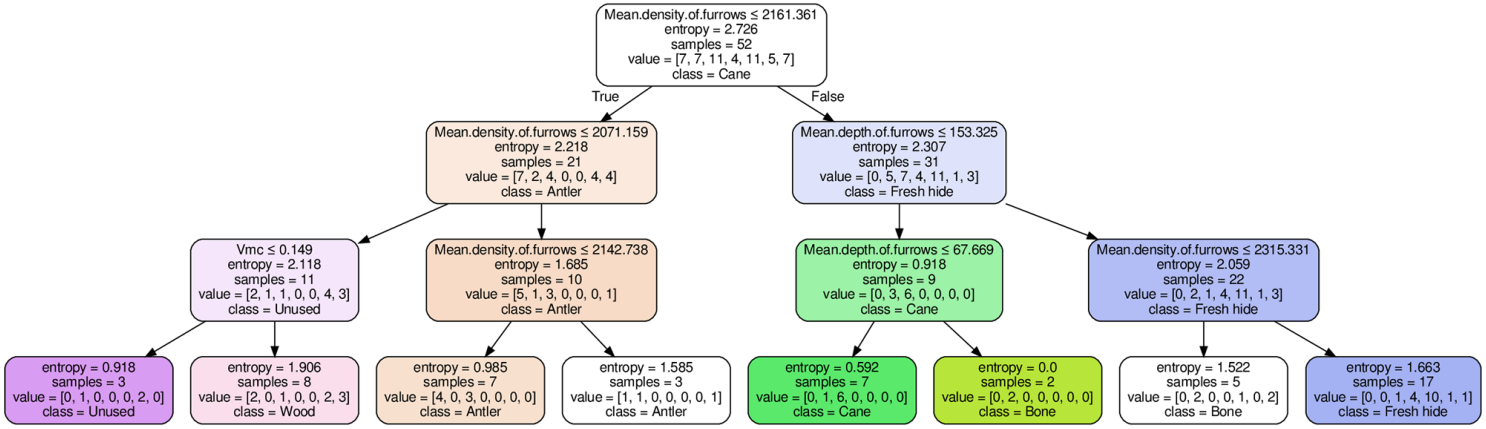


Figure B.2.: Learned decision tree without considering the quartzite variety (type) and only the selected surface parameters as features. The attributes in each box are as follows: First, the feature considered and the corresponding inequality, second the entropy of the splitting and third the number of samples in the node. The fourth line shows a list of samples per class in the node, where classes are numbered as 0 = Antler, 1 = Bone, 2 = Cane, 3 = Dry hide, 4 = Fresh hide, 5 = Unused, 6 = Wood. The last line shows finally the class with most members in the sample, which is used as a label for all data points in that node. Image taken from [158, Fig. 4].

Support vector machine

As another more advanced approach, support vector machines (SVM) [87, chapter 12] [44] [186] were used, especially to check whether the results from the decision tree are bounded by model complexity.

Concept

The basic idea of the SVM is to separate points of the two classes in a binary classification problem by hyperplanes in the feature space, which are chosen to have maximum distance (margin) to the data points of each class. Thus, the location of those hyperplanes is determined by some features only, called support vectors and giving the name to the method, see Figure B.3 for an illustration. By choosing the hyperplanes in this way one attempts to minimize the generalization error.

Following the presentation [87, chapter 12], the mathematical formulation of the SVM for two classes can be given as follows: the hyperplane of interest can described as the set of all vectors $\boldsymbol{x}$ that satisfy the condition

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle - b = 0, \tag{B.6}$$

where $\boldsymbol{w}$ is a vector normal to that plane and b a scalar. When considering the support vectors $\boldsymbol{x}_\pm$ as those residing on the hyperplanes

$$\langle \boldsymbol{w}, \boldsymbol{x}_\pm \rangle - b = \pm 1 \tag{B.7}$$

the size of the margin s can be computed as the projection of the difference between the support vectors onto a unit length normal vector of the hyperplane, i.e.

$$s = \left\langle (\boldsymbol{x}_+ - \boldsymbol{x}_-), \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right\rangle = \frac{\langle \boldsymbol{w}, \boldsymbol{x}_+ \rangle - \langle \boldsymbol{w}, \boldsymbol{x}_- \rangle}{\|\boldsymbol{w}\|} = \frac{1 + b - (-1 + b)}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}. \tag{B.8}$$

Generalizing from equation (B.7), for a data point in question that is described by a feature vector $\boldsymbol{x}_f$, the function

$$c(\boldsymbol{x}_f) = \operatorname{sgn}\left(\langle \boldsymbol{w}, \boldsymbol{x}_f \rangle - b \right) \tag{B.9}$$

gives the estimated class label as -1 or 1.

For two non-overlapping classes the parameters $\boldsymbol{w}$ and b can be computed by maximizing s in equation (B.8) under the constraints that $c(\cdot)$ from equation (B.9) gives the correct label for all training examples. However, it can be shown that this is equivalent to minimizing the labeling errors counted by the loss function

$$l(\mathbf{w}, \lambda) = \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b)\right), \tag{B.10}$$

where $\boldsymbol{x}_i$ denotes the feature vector and $y_i \in \{-1, 1\}$ the label of data point i. For the details of the minimization see the discussion in [87, chapter 12].

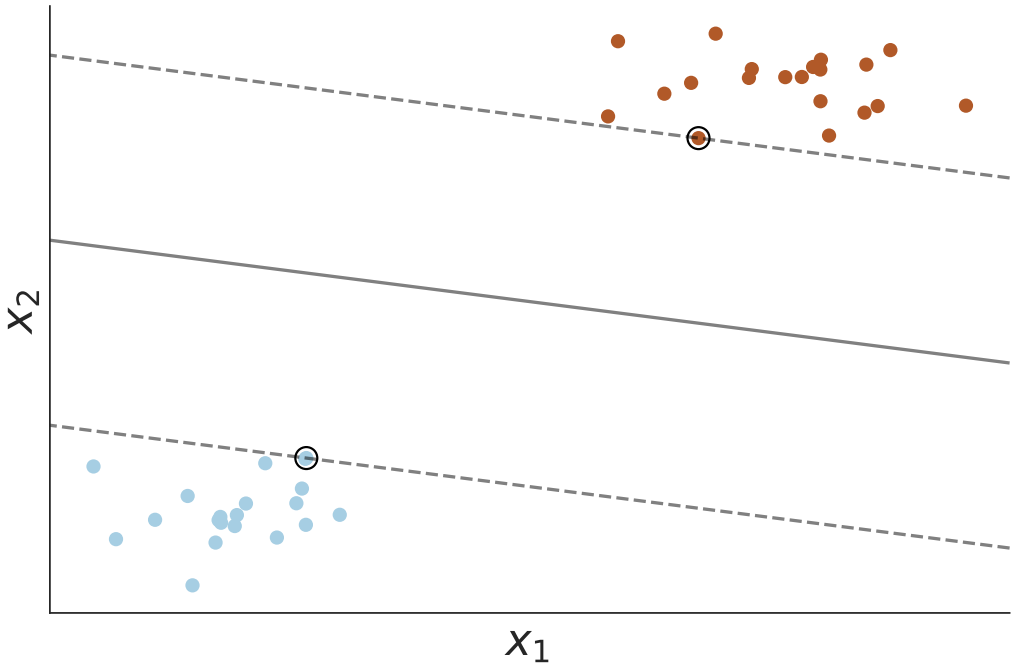


Figure B.3.: Concept of a support vector machine.
For ease of illustration the feature space has only two dimensions, x_1 and x_2. The color of the dots encodes the true class labels, the support vectors $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ are marked with additional black circles. The solid line marks the hyperplane in equation (B.6), the dashed lines the margins in equation (B.7).
Recreated after [87, Fig. 12.1].

If there are classes that cannot be linearly separated, there are two ways to cope with this problem: first, the loss function in equation (B.10) can be extended by another term

$$l(\mathbf{w}, \lambda) = \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b)\right) + \lambda \|\mathbf{w}\|^2, \tag{B.11}$$

such that one still tries to minimize the labeling error while also having a large margin. The regularization parameter λ controls the trade-off between the two goals.

The second approach is known as the so-called kernel trick: consider a map $\phi : F \to S, \boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ from the feature space F into a space S with more dimensions, i.e. $\dim(S) > \dim(F)$. When using a suitable map ϕ, the classes are linearly separable in S. As for the loss functions (B.10) and (B.11) only the inner products $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$ need to be computed, it is beneficial to choose maps ϕ such that the inner products $\langle \phi(\boldsymbol{w}), \phi(\boldsymbol{x}_i) \rangle$ in S can be computed as a function of the untransformed vectors

$$\langle \phi(\boldsymbol{w}), \phi(\boldsymbol{x}_i) \rangle = K(\boldsymbol{w}, \boldsymbol{x}_i), \tag{B.12}$$

where $K(\cdot, \cdot)$ is called the kernel function. In practice, instead of specifying $\phi(\cdot)$ usually the kernel function is chosen. Popular kernel functions are

$$\begin{aligned} K_{\mathrm{Poly}}(\boldsymbol{w}, \boldsymbol{x}_i) &= (\gamma \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + r)^d \\ K_{\mathrm{RB}}(\boldsymbol{w}, \boldsymbol{x}_i) &= \exp(-\gamma \|\boldsymbol{w} - \boldsymbol{x}_i\|^2) \\ K_{\mathrm{NN}}(\boldsymbol{w}, \boldsymbol{x}_i) &= \tanh(\gamma \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + r), \end{aligned}$$

which are often called the polynomial, radial basis and neural network kernels [87, section 12.3.1]. The resulting classification functions

$$c_\phi(\boldsymbol{x}_f) = \operatorname{sgn}\left(\langle \phi(\boldsymbol{w}), \phi(\boldsymbol{x}_i) \rangle - b\right) = \operatorname{sgn}\left(K(\boldsymbol{w}, \boldsymbol{x}_i) - b\right) \tag{B.13}$$

now allow for nonlinear hypersurfaces in the feature space F.

Computation

For the actual computation the implementation in the *scikit-learn* package [159] was used, which in turn uses the libary of Chang and Lin [38]. As the task at hand involved more than two classes, the so-called one-versus-one approach is used, where for each pair of classes a SVM is trained and the most dominant class label is chosen [87, section 12.3.8].

Discussion

As described above, the SVM is able to use non-linear separation functions, which is a clear improvement over the decision tree. While the considerably greater computational effort is not a problem on the small dataset, the greater number of hyperparameters (kernel functions with coefficients and regularization parameter λ) might be too high for the usage with small data sets like those used here. Moreover, the working principle is already largely nontransparent for the general audience.

Training

Data split

First, the data was split into a test (33% of data, n = 26 for the full dataset and n = 24 for the restricted dataset) and a training set (remaining data, n = 52 for the full dataset and n = 46 for the restricted dataset) in a stratified fashion, such that quartzite types and labels were approximately equally present in both test and training data.

Hyperparameter tuning

Then 3-fold cross-validation using balanced accuracy [105] for measurement of the performance was used to tune the respective hyperparameters. Finally, the algorithms with the estimated optimal hyperparameter setting were run once on the on full training data to produce the versions for evaluation.

This whole procedure was performed once using the $k = 33$ parameters and the quartzite type as features, and once using the $k = 33$ parameters without the quartzite type as a feature.

B.4. Results

On the test data the following results were achieved:

Difference between datasets

Concerning the overall picture, the results on the full and restricted data sets are quite similar, but do differ in details like the selected features and classification rates. The detailed results are available at https://doi.org/10.5281/zenodo.4249219, such that in the following the presentation is restricted to the full dataset, where generally the classification rates are higher, thus being more promising to study.

Feature selection and quartzite variety

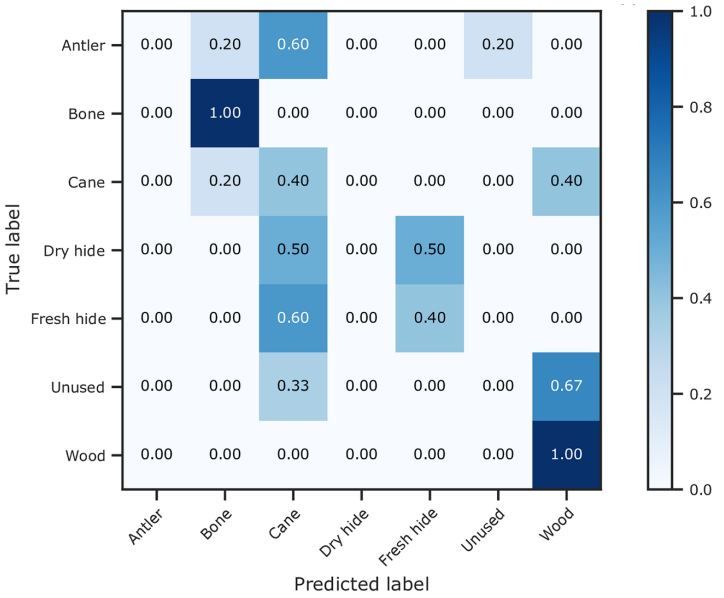
It turned out that using only the selected or all surface parameters and using or not using the quartzite variety resulted in comparable classification rates. Thus for a better practicability and to avoid overfitting, the selected surface parameters can be used as features only. Those selected are namely core material volume (Vmc), mean depth of furrows and mean density of furrows.

Decision tree and SVM

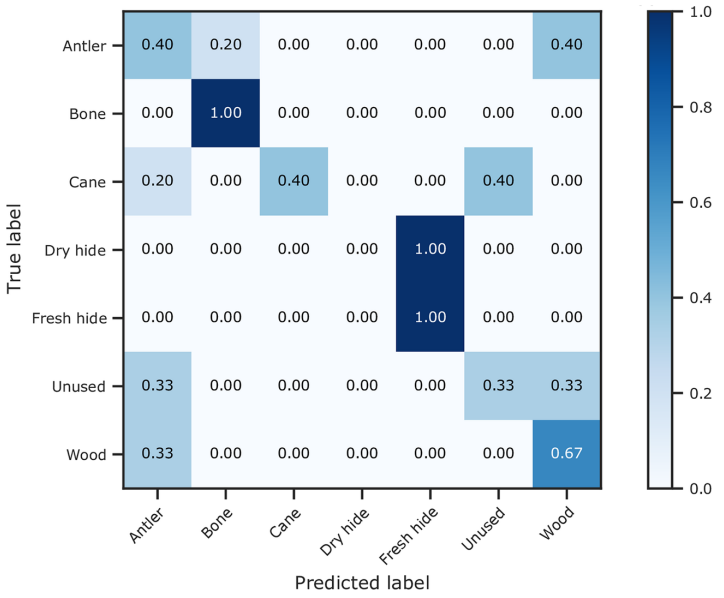
The difference in performance between the decision tree and the SVM was not significant enough such that it is favorable to use the decision tree due to its easier interpretability and less complexity that in turn prevents overfitting.

Confusion matrix

The results of the classification task using the decision tree on the full data set are shown in Figure B.4.



(a) Using the quartzite variety as a feature.



(b) Without using the quartzite variety as a feature.

Figure B.4.: Confusion matrix for the decision tree on the test with and without using the quartzite variety as a feature. Each cell shows the fraction of cases in which the pair of corresponding predicated and true label occured, normed for the true labels. Recreated after [158, Fig. 3].

Bibliography

[1] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/ (cit. on pp. 42, 43, 107).

[2] R. Aebersold and M. Mann. "Mass spectrometry-based proteomics". In: *Nature* 422.6928 (Mar. 2003), pp. 198–207. ISSN: 0028-0836. DOI: 10.1038/nature01511 (cit. on p. 97).

[3] V. V. Albani, R. M. Velho, and J. P. Zubelli. "Estimating, monitoring, and forecasting COVID-19 epidemics: a spatiotemporal approach applied to NYC data". In: *Scientific Reports* 11.1 (Dec. 2021), pp. 1–15. ISSN: 20452322. arXiv: 2011.08664. DOI: 10.1038/s41598-021-88281-w (cit. on pp. 71, 73, 77).

[4] R. Albert and A.-L. Barabási. "Statistical mechanics of complex networks". In: *Reviews of Modern Physics* 74.1 (Jan. 2002), pp. 47–97. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.74.47 (cit. on p. 71).

[5] N. L. Anderson and N. G. Anderson. "Proteome and proteomics: New technologies, new concepts, and new words". In: *Electrophoresis* 19.11 (Aug. 1998), pp. 1853–1861. ISSN: 0173-0835. DOI: 10.1002/elps.1150191103 (cit. on p. 97).

[6] T. Arakaki. *JuliaFolds/Folds.jl: A unified interface for sequential, threaded, and distributed fold*. URL: https://github.com/JuliaFolds/Folds.jl (cit. on p. 35).

[7] E. S. Baker et al. "An LC-IMS-MS Platform Providing Increased Dynamic Range for High-Throughput Proteomic Studies". In: *Journal of Proteome Research* 9.2 (Feb. 2010), pp. 997–1006. ISSN: 1535-3893. DOI: 10.1021/pr900888b (cit. on p. 98).

[8] A. Barp et al. "A Unifying and Canonical Description of Measure-Preserving Diffusions". May 2021. URL: http://arxiv.org/abs/2105.02845 (cit. on p. 26).

[9] D. Basu. "On the Elimination of Nuisance Parameters". In: *Selected Works of Debabrata Basu*. Springer New York, 2011, pp. 279–290. DOI: 10.1007/978-1-4419-5825-9_26 (cit. on p. 10).

[10] C. Bauer, R. Cramer, and J. Schuchhardt. "Evaluation of peak-picking algorithms for protein mass spectrometry." In: *Methods in molecular biology (Clifton, N.J.)* 696 (2011), pp. 341–352. ISSN: 19406029. DOI: 10.1007/978-1-60761-987-1_22 (cit. on p. 111).

[11] T. Bayes. "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S". In: *Philosophical Transactions of the Royal Society of London* 53 (Dec. 1763), pp. 370–418. ISSN: 0261-0523. DOI: 10.1098/rstl.1763.0053 (cit. on p. 8).

[12] V. Belik, T. Geisel, and D. Brockmann. "Natural Human Mobility Patterns and Spatial Spread of Infectious Diseases". In: *Physical Review X* 1.1 (Aug. 2011), p. 011001. issn: 2160-3308. doi: 10.1103/PhysRevX.1.011001 (cit. on p. 71).

[13] K. Berlin et al. "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing". In: *Nature Biotechnology* 33.6 (June 2015), pp. 623–630. issn: 1087-0156. doi: 10.1038/nbt.3238 (cit. on p. 101).

[14] N. Best, S. Richardson, and A. Thomson. *A comparison of Bayesian spatial models for disease mapping*. Feb. 2005. doi: 10.1191/0962280205sm388oa (cit. on p. 71).

[15] M. J. Betancourt, S. Byrne, and M. Girolami. "Optimizing The Integrator Step Size for Hamiltonian Monte Carlo". Nov. 2014. url: http://arxiv.org/abs/1411.6669 (cit. on p. 26).

[16] M. J. Betancourt and M. Girolami. "Hamiltonian Monte Carlo for Hierarchical Models". In: *Current Trends in Bayesian Methodology with Applications* (Dec. 2013), pp. 79–101. url: http://arxiv.org/abs/1312.0906 (cit. on p. 24).

[17] M. Betancourt. "A Short Review of Ergodicity and Convergence of Markov chain Monte Carlo Estimators". Oct. 2021. url: http://arxiv.org/abs/2110.07032 (cit. on p. 22).

[18] M. Betancourt. "Diagnosing Suboptimal Cotangent Disintegrations in Hamiltonian Monte Carlo". Apr. 2016. url: http://arxiv.org/abs/1604.00695 (cit. on p. 27).

[19] M. Betancourt et al. "The geometric foundations of Hamiltonian Monte Carlo". In: *Bernoulli* 23.4A (Nov. 2017), pp. 2257–2298. issn: 1350-7265. doi: 10.3150/16-BEJ810 (cit. on p. 26).

[20] J. Bezanson et al. "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1 (Jan. 2017), pp. 65–98. issn: 0036-1445. doi: 10.1137/141000671 (cit. on pp. 35, 41, 46, 83).

[21] J. Bhandari Neupane et al. "Characterization of Leptazolines A–D, Polar Oxazolines from the Cyanobacterium Leptolyngbya sp., Reveals a Glitch with the "WilloughbyHoye" Scripts for Calculating NMR Chemical Shifts". In: *Organic Letters* 21.20 (2019), pp. 8449–8453. doi: 10.1021/acs.orglett.9b03216 (cit. on p. 88).

[22] P. J. Birrell et al. "Efficient real-time monitoring of an emerging influenza pandemic: How feasible?" In: *The Annals of Applied Statistics* 14.1 (Mar. 2020), pp. 74–93. issn: 1932-6157. doi: 10.1214/19-AOAS1278 (cit. on p. 73).

[23] W. P. Blackstock and M. P. Weir. "Proteomics: quantitative and physical mapping of cellular proteins". In: *Trends in Biotechnology* 17.3 (Mar. 1999), pp. 121–127. issn: 0167-7799. doi: 10.1016/S0167-7799(98)01245-1 (cit. on p. 97).

[24] K. Bob, B. Schmidt, and A. Hildebrandt. "Extending Turing.jl for Online Updates of Bayesian Inference in Julia" (cit. on p. 45).

[25] K. Bob et al. "Locality-sensitive hashing enables efficient and scalable signal classification in high-throughput mass spectrometry raw data". In: *Submitted to BMC Bioinformatics* (2021) (cit. on p. 97).

[26] T. Bollerslev. “Generalized autoregressive conditional heteroskedasticity”. In: *Journal of Econometrics* 31.3 (Apr. 1986), pp. 307–327. issn: 03044076. doi: 10.1016/0304-4076(86)90063-1 (cit. on p. 48).

[27] J. Bracher et al. “Evaluating epidemic forecasts in an interval format”. In: *PLOS Computational Biology* 17.2 (Feb. 2021). Ed. by V. E. Pitzer, e1008618. issn: 1553-7358. doi: 10.1371/journal.pcbi.1008618 (cit. on pp. 73, 82).

[28] D. Brockmann and D. Helbing. “The hidden geometry of complex, network-driven contagion phenomena”. In: *Science* 342.6164 (Dec. 2013), pp. 1337–1342. issn: 10959203. doi: 10.1126/science.1245200 (cit. on p. 71).

[29] C. A. Brown et al. “Multiscale analyses and characterizations of surface topographies”. In: *CIRP Annals* 67.2 (Jan. 2018), pp. 839–862. issn: 17260604. doi: 10.1016/j.cirp.2018.06.001 (cit. on p. 57).

[30] A. Bryant. “A Bayesian Computation Graph for High-Performance Gradient Evaluation on the JVM”. In: *International Conference on Probabilistic Programming (PROBPROG) Oct 22 - 24 2020*. 2020 (cit. on pp. 42, 43).

[31] A. Buchholz, N. Chopin, and P. E. Jacob. “Adaptive Tuning of Hamiltonian Monte Carlo Within Sequential Monte Carlo”. In: *Bayesian Analysis* 16.3 (Sept. 2021), pp. 745–771. issn: 1936-0975. doi: 10.1214/20-BA1222 (cit. on p. 28).

[32] I. Calandra et al. “The effect of numerical aperture on quantitative use-wear studies and its implication on reproducibility”. In: *Scientific Reports* 9.1 (2019). issn: 20452322. doi: 10.1038/s41598-019-42713-w (cit. on pp. 28, 57, 58, 62, 64, 65, 67, 68).

[33] I. Calandra et al. “Dental microwear texture analysis in Toothfrax and MountainsMap SSFA module: Different software packages, different results? [pre-print]”. In: (Apr. 2021). doi: 10.5281/ZENODO.4671439 (cit. on pp. 57, 58, 64–67).

[34] J. V. Candy. *Bayesian Signal Processing: Classical, Modern, and Particle Filtering Methods: Second Edition*. Hoboken, NJ, USA: Wiley Blackwell, Aug. 2016, pp. 1–601. isbn: 9781119125495. doi: 10.1002/9781119125495 (cit. on pp. 28, 45).

[35] L. Cardelli. “Type Systems”. In: *Computer Science Handbook*. Ed. by A. B. Tucker. Second Edi. Taylor and Francis, 2004. isbn: 9781584883609 (cit. on p. 42).

[36] B. Carpenter. “What do we need from a probabilistic programming language to support Bayesian workflow?” In: *International Conference on Probabilistic Programming (PROBPROG)*. 2021 (cit. on p. 46).

[37] B. Carpenter et al. “Stan: A probabilistic programming language”. In: *Journal of statistical software* 76.1 (2017). url: https://doi.org/10.18637/jss.v076.i01 (cit. on pp. 41, 42, 46, 82, 84, 85).

[38] C. C. Chang and C. J. Lin. “LIBSVM: A Library for support vector machines”. In: *ACM Transactions on Intelligent Systems and Technology* 2.3 (Apr. 2011), pp. 1–27. issn: 21576904. doi: 10.1145/1961189.1961199 (cit. on p. 122).

[39] S. Chang et al. “Mobility network models of COVID-19 explain inequities and inform reopening”. In: *Nature* 589.7840 (Jan. 2021), pp. 82–87. issn: 0028-0836. doi: 10.1038/s41586-020-2923-3 (cit. on p. 71).

[40] M. S. Charikar and M. S. "Similarity estimation techniques from rounding algorithms". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02*. New York, New York, USA: ACM Press, 2002, p. 380. ISBN: 1581134959. DOI: 10.1145/509907.509965 (cit. on p. 111).

[41] *CmdStan.jl*. URL: https://github.com/StanJulia/CmdStan.jl (cit. on p. 85).

[42] M. L. Cohen. "Changing patterns of infectious disease". In: *Nature* 406.6797 (Aug. 2000), pp. 762–767. ISSN: 0028-0836. DOI: 10.1038/35021206 (cit. on p. 69).

[43] C. E. Cook et al. "The European Bioinformatics Institute in 2018: tools, infrastructure and training". In: *Nucleic Acids Research* 47.D1 (Jan. 2019), pp. D15–D22. ISSN: 0305-1048. DOI: 10.1093/nar/gky1124 (cit. on p. 100).

[44] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 0885-6125. DOI: 10.1007/bf00994018 (cit. on p. 120).

[45] J. Cox and M. Mann. "MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification". In: *Nature Biotechnology* 26.12 (Dec. 2008), pp. 1367–1372. ISSN: 10870156. DOI: 10.1038/nbt.1511 (cit. on p. 101).

[46] D. Crisan, J. Míguez, and G. Ríos-Muñoz. "On the performance of parallelisation schemes for particle filtering". In: *Eurasip Journal on Advances in Signal Processing* 2018.1 (Dec. 2018), p. 31. ISSN: 16876180. DOI: 10.1186/s13634-018-0552-x (cit. on p. 30).

[47] M. F. Cusumano-Towner. "Gen: A High-Level Programming Platform for Probabilistic Inference". PhD thesis. 2020. URL: https://hdl.handle.net/1721.1/129247 (cit. on pp. 42, 43).

[48] J. Dean and S. Ghemawat. "MapReduce: Simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 00010782. DOI: 10.1145/1327452.1327492 (cit. on p. 35).

[49] J. Dehning et al. "Inferring change points in the spread of COVID-19 reveals the effectiveness of interventions". In: *Science* 369.6500 (July 2020). ISSN: 10959203. DOI: 10.1126/science.abb9789 (cit. on pp. 70, 73, 77).

[50] P. Del Moral and L. Miclo. "Branching and interacting particle systems approximations of Feynman-Kac formulae with applications to non-linear filtering". In: *Séminaire de probabilités de Strasbourg, Volume 34 (2000)*. Springer, Berlin, Heidelberg, 2000, pp. 1–145. DOI: 10.1007/bfb0103798 (cit. on p. 28).

[51] P. Del Moral, A. Doucet, and A. Jasra. "Sequential Monte Carlo samplers". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.3 (June 2006), pp. 411–436. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2006.00553.x (cit. on pp. 28, 45, 52).

[52] O. Diekmann, H. Heesterbeek, and T. Britton. *Mathematical Tools for Understanding Infectious Disease Dynamics*. Princeton University Press, Dec. 2012. ISBN: 978-1-4008-4562-0. DOI: 10.1515/9781400845620 (cit. on p. 70).

[53] J. V. Dillon et al. "TensorFlow Distributions". Nov. 2017. URL: http://arxiv.org/abs/1711.10604 (cit. on p. 42).

[54] J. Ding, V. Tarokh, and Y. Yang. “Model Selection Techniques: An Overview”. In: *IEEE Signal Processing Magazine* 35.6 (Nov. 2018), pp. 16–34. issn: 15580792. arXiv: 1810.09583. doi: 10.1109/MSP.2018.2867638 (cit. on p. 6).

[55] U. Distler et al. “Drift time-specific collision energies enable deep-coverage data-independent acquisition proteomics”. In: *Nature Methods* 11.2 (Feb. 2014), pp. 167–170. issn: 1548-7091. doi: 10.1038/nmeth.2767 (cit. on p. 98).

[56] U. Distler et al. “Label-free quantification in ion mobility-enhanced data-independent acquisition proteomics”. In: *Nature Protocols* 11.4 (Apr. 2016), pp. 795–812. issn: 17502799. doi: 10.1038/nprot.2016.042 (cit. on p. 113).

[57] T. Donker et al. “Nowcasting pandemic influenza A/H1N1 2009 hospitalizations in the Netherlands”. In: *European Journal of Epidemiology* 26.3 (Mar. 2011), pp. 195–201. issn: 03932990. doi: 10.1007/s10654-011-9566-5 (cit. on p. 72).

[58] S. Duane et al. “Hybrid Monte Carlo”. In: *Physics Letters B* 195.2 (Sept. 1987), pp. 216–222. issn: 03702693. doi: 10.1016/0370-2693(87)91197-X (cit. on p. 24).

[59] R. Durbin et al. *Biological Sequence Analysis - Probabilistic Models of Proteins and Nucleic Acids*. Cambridge: Cambridge University Press, 1998. isbn: 978-1-139-45739-2 (cit. on p. 21).

[60] D. Dutta and T. Chen. “Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search”. In: *Bioinformatics* 23.5 (Mar. 2007), pp. 612–618. issn: 1367-4803. doi: 10.1093/bioinformatics/btl645 (cit. on p. 101).

[61] D. J. Earl and M. W. Deem. *Parallel tempering: Theory, applications, and new perspectives*. Dec. 2005. arXiv: 0508111 [physics]. doi: 10.1039/b509983h (cit. on p. 34).

[62] M. Egger, O. Razum, and A. Rieder, eds. *Public Health Kompakt*. De Gruyter, Mar. 2021. isbn: 9783110673708. doi: 10.1515/9783110673708 (cit. on p. 69).

[63] A. Einstein. “Zur Elektrodynamik bewegter Körper”. In: *Annalen der Physik* 322.10 (Jan. 1905), pp. 891–921. issn: 00033804. doi: 10.1002/andp.19053221004 (cit. on p. 6).

[64] M. Falcioni and M. W. Deem. “A biased Monte Carlo scheme for zeolite structure solution”. In: *Journal of Chemical Physics* 110.3 (Jan. 1999), pp. 1754–1766. issn: 00219606. doi: 10.1063/1.477812 (cit. on p. 34).

[65] E. F. Fama and K. R. French. “Common risk factors in the returns on stocks and bonds”. In: *Journal of Financial Economics* 33.1 (Feb. 1993), pp. 3–56. issn: 0304405X. doi: 10.1016/0304-405X(93)90023-5 (cit. on p. 60).

[66] A. Fick. *Philosophischer Versuch über die Wahrscheinlichkeiten*. Würzburg: Druck und Verlag der Stahel’schen Univers.-buch. & kunsthandlung, 1883, 50 p. url: http://hdl.handle.net/2027/mdp.39015022449592 (cit. on p. 7).

[67] D. Fink. “A Compendium of Conjugate Priors”. 1997. url: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.157.5540 (cit. on p. 15).

[68] J. A. Firth et al. “Using a real-world network to model localized COVID-19 control strategies”. In: *Nature Medicine* 26.10 (Oct. 2020), pp. 1616–1622. issn: 1546170X. doi: 10.1038/s41591-020-1036-8 (cit. on p. 71).

[69] W. Fox Smith. *Experimental Physics Principles and Practice for the Laboratory*. Ed. by Walter Fox Smith. Taylor and Francis, 2020, p. 451. isbn: 9781498778473 (cit. on p. 6).

[70] H. G. Gauch Jr. *Scientific Method in Practice*. Cambridge University Press, Dec. 2002. isbn: 9780521816892. doi: 10.1017/CBO9780511815034 (cit. on p. 6).

[71] H. Ge, K. Xu, and Z. Ghahramani. "Turing: a language for flexible probabilistic inference". In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*. 2018, pp. 1682–1690. url: http://proceedings.mlr.press/v84/ge18b.html (cit. on pp. 35, 41–43, 46, 82, 84, 85).

[72] A. H. Gebremedhin and A. Walther. "An introduction to algorithmic differentiation". In: *WIREs Data Mining and Knowledge Discovery* 10.1 (Jan. 2020), e1334. issn: 1942-4787. doi: 10.1002/widm.1334 (cit. on p. 42).

[73] A. Gelman and D. B. Rubin. "Inference from Iterative Simulation Using Multiple Sequences". In: *Statistical Science* 7.4 (Nov. 1992), pp. 457–472. issn: 0883-4237. doi: 10.1214/ss/1177011136 (cit. on p. 26).

[74] A. Gelman et al. *Bayesian Data Analysis*. Third Edition. Boca Raton, Fla: CRC Press, 2013. isbn: 978-1-439-84095-5 (cit. on pp. 15, 18, 23, 26–28, 63).

[75] A. Gelman et al. "Bayesian Workflow". Nov. 2020. url: http://arxiv.org/abs/2011.01808 (cit. on pp. 12, 13, 63).

[76] S. Geman and D. Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (1984), pp. 721–741. issn: 01628828. doi: 10.1109/TPAMI.1984.4767596 (cit. on p. 23).

[77] A. Gionis, P. Indyk, and R. Motwani. "Similarity Search in High Dimensions via Hashing". In: *Proceedings of the 25th International Conference on Very Large Data Bases* (1999), pp. 518–529. url: https://dl.acm.org/citation.cfm?id=671516 (cit. on p. 101).

[78] T. Gneiting, F. Balabdaoui, and A. E. Raftery. "Probabilistic forecasts, calibration and sharpness". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69.2 (Apr. 2007), pp. 243–268. issn: 1369-7412. doi: 10.1111/j.1467-9868.2007.00587.x (cit. on pp. 73, 82).

[79] T. Gneiting and A. E. Raftery. "Strictly Proper Scoring Rules, Prediction, and Estimation". In: *Journal of the American Statistical Association* 102.477 (Mar. 2007), pp. 359–378. issn: 0162-1459. doi: 10.1198/016214506000001437 (cit. on pp. 51, 73, 82).

[80] N. C. Grassly and C. Fraser. *Mathematical models of infectious disease transmission*. June 2008. doi: 10.1038/nrmicro1845 (cit. on pp. 69, 71).

[81] J. L. Green and D. A. Croft. "Using dental mesowear and microwear for dietary inference: A review of current techniques and applications". In: *Vertebrate Paleobiology and Paleoanthropology*. Springer, 2018, pp. 53–73. doi: 10.1007/978-3-319-94265-0_5 (cit. on p. 57).

[82] J. Griss et al. “Recognizing millions of consistently unidentified spectra across hundreds of shotgun proteomics datasets”. In: *Nature Methods* 13.8 (July 2016), pp. 651–656. issn: 15487105. doi: 10.1038/nmeth.3902 (cit. on p. 98).

[83] F. Günther et al. “Nowcasting the COVID-19 pandemic in Bavaria”. In: *Biometrical Journal* 63.3 (2020), pp. 490–502. doi: https://doi.org/10.1002/bimj.202000112 (cit. on pp. 72, 73, 76, 77, 81, 82, 85).

[84] M. Hanke-Bourgeois. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Berlin Heidelberg New York: Springer-Verlag, 2009. isbn: 978-3-834-89309-3 (cit. on p. 18).

[85] C. R. Harris et al. *Array programming with NumPy*. Sept. 2020. arXiv: 2006.10256. doi: 10.1038/s41586-020-2649-2 (cit. on p. 43).

[86] HashiCorps. *Terraform*. url: https://www.terraform.io/ (cit. on p. 89).

[87] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Berlin Heidelberg: Springer Science & Business Media, 2013. isbn: 978-0-387-21606-5 (cit. on pp. 118, 120–122).

[88] W. K. Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: *Biometrika* 57.1 (Apr. 1970), pp. 97–109. issn: 1464-3510. doi: 10.1093/biomet/57.1.97 (cit. on p. 23).

[89] A.-K. Hildebrandt et al. “CorCast: A Distributed Architecture for Bayesian Epidemic Nowcasting and its Application to District-Level SARS-CoV-2 Infection Numbers in Germany”. In: *medRxiv* (June 2021). doi: 10.1101/2021.06.02.21258209 (cit. on p. 75).

[90] M. Höhle. “Surveillance: An R package for the monitoring of infectious diseases”. In: *Computational Statistics* 22.4 (2007), pp. 571–582 (cit. on p. 77).

[91] M. Höhle and M. an der Heiden. “Bayesian nowcasting during the STEC O104:H4 outbreak in Germany, 2011”. In: *Biometrics* 70.4 (2014), pp. 993–1002. doi: https://doi.org/10.1111/biom.12194 (cit. on pp. 72, 73, 76, 77, 81, 85).

[92] A. J. Holbrook et al. “Massive Parallelization Boosts Big Bayesian Multidimensional Scaling”. In: *Journal of Computational and Graphical Statistics* 30.1 (Jan. 2021), pp. 11–24. issn: 1061-8600. doi: 10.1080/10618600.2020.1754226 (cit. on pp. 30, 33, 73).

[93] M. D. Homan and A. Gelman. “The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo”. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1593–1623. issn: 1532-4435. url: http://dl.acm.org/citation.cfm?id=2627435.2638586 (cit. on pp. 26, 28, 84, 85).

[94] M. B. Hooten, D. S. Johnson, and B. M. Brost. “Making Recursive Bayesian Inference Accessible”. In: *The American Statistician* 75.2 (July 2018), pp. 185–194. issn: 0003-1305. doi: 10.1080/00031305.2019.1665584 (cit. on p. 30).

[95] L. Hufnagel, D. Brockmann, and T. Geisel. “Forecast and control of epidemics in a globalized world”. In: *Proceedings of the National Academy of Sciences of the United States of America* 101.42 (Oct. 2004), pp. 15124–15129. issn: 00278424. doi: 10.1073/pnas.0308344101 (cit. on p. 71).

[96] K. Hukushima and K. Nemoto. "Exchange Monte Carlo Method and Application to Spin Glass Simulations". In: *Journal of the Physical Society of Japan* 65.6 (Nov. 1996), pp. 1604–1608. ISSN: 00319015. arXiv: 9512035 [cond-mat]. DOI: 10.1143/JPSJ.65.1604 (cit. on p. 34).

[97] J. C. Hull. *Options, Futures, and Other Derivatives*. Ninth Edition. Amsterdam: Pearson Education, 2018. ISBN: 978-1-292-21289-0 (cit. on p. 21).

[98] P. Indyk and R. Motwani. "Approximate nearest neighbors". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*. New York, New York, USA: ACM Press, 1998, pp. 604–613. ISBN: 0897919629. DOI: 10.1145/276698.276876 (cit. on p. 101).

[99] International Organization for Standardization. *ISO 25178 –2– Geometrical product specifications (GPS)–Surface texture: Areal—Part 2: Terms, definitions and surface texture parameters*. 2012 (cit. on pp. 58, 59).

[100] W. Jakob, J. Rhinelander, and D. Moldovan. *pybind11 – Seamless operability between C++11 and Python*. 2017 (cit. on p. 107).

[101] E. T. Jaynes. *Probability Theory - The Logic of Science*. Cambridge: Cambridge University Press, 2003. ISBN: 978-0-521-59271-0 (cit. on pp. 7–10, 63).

[102] E. T. Jaynes. "Information theory and statistical mechanics". In: *Physical Review* 106.4 (May 1957), pp. 620–630. ISSN: 0031899X. DOI: 10.1103/PhysRev.106.620 (cit. on p. 63).

[103] E. T. Jaynes. "Information theory and statistical mechanics. II". In: *Physical Review* 108.2 (Oct. 1957), pp. 171–190. ISSN: 0031899X. DOI: 10.1103/PhysRev.108.171 (cit. on p. 63).

[104] M. J. Keeling and K. T. Eames. "Networks and epidemic models". In: *Journal of The Royal Society Interface* 2.4 (Sept. 2005), pp. 295–307. ISSN: 1742-5689. DOI: 10.1098/rsif.2005.0051 (cit. on p. 71).

[105] J. D. Kelleher, B. M. Namee, and A. D'Arcy. *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. Cambridge: MIT Press, 2015. ISBN: 978-0-262-04469-1 (cit. on p. 123).

[106] D. E. Knuth. "An algorithm for Brownian zeroes". In: *Computing* 33.1 (Mar. 1984), pp. 89–94. ISSN: 0010485X. DOI: 10.1007/BF02243079 (cit. on p. 21).

[107] A. Kraskov, H. Stögbauer, and P. Grassberger. "Estimating mutual information". In: *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 69.6 (June 2004), p. 16. ISSN: 1063651X. arXiv: 0305641 [cond-mat]. DOI: 10.1103/PhysRevE.69.066138 (cit. on pp. 116, 117).

[108] M. Kröger and R. Schlickeiser. "Analytical solution of the SIR-model for the temporal evolution of epidemics. Part A: time-independent reproduction factor". In: *Journal of Physics A: Mathematical and Theoretical* 53.50 (Dec. 2020), p. 505601. ISSN: 17518121. DOI: 10.1088/1751-8121/abc65d (cit. on p. 70).

[109] J. K. Kruschke. *Doing Bayesian Data Analysis - A Tutorial with R, JAGS, and Stan*. Second Edi. Amsterdam, Boston: Academic Press, 2015. ISBN: 978-0-124-05888-0 (cit. on p. 15).

[110] M. K. Łącki, D. Valkenborg, and M. P. Startek. "IsoSpec2: Ultrafast Fine Structure Calculator". In: *Analytical Chemistry* 92.14 (July 2020), pp. 9472–9475. issn: 15206882. doi: 10.1021/acs.analchem.0c00959 (cit. on p. 98).

[111] M. K. Łącki et al. "OpenTIMS, TimsPy, and TimsR: Open and Easy Access to timsTOF Raw Data". In: *Journal of Proteome Research* 20.4 (Apr. 2021), pp. 2122–2129. issn: 15353907. doi: 10.1021/acs.jproteome.0c00962 (cit. on pp. 107, 113).

[112] R. Lämmel. "Google's MapReduce programming model - Revisited". In: *Science of Computer Programming* 68.3 (Oct. 2007), pp. 208–237. issn: 01676423. doi: 10.1016/j.scico.2007.07.001 (cit. on p. 35).

[113] A. B. Lawson. *Using R for Bayesian Spatial and Spatio-Temporal Health Modeling*. Chapman and Hall/CRC, Apr. 2021. isbn: 9781003043997. doi: 10.1201/9781003043997 (cit. on p. 73).

[114] J. Leskovec et al. "Kronecker Graphs: An Approach to Modeling Networks". In: *Journal of Machine Learning Research* 11 (Dec. 2008), pp. 985–1042. url: http://arxiv.org/abs/0812.4905 (cit. on p. 71).

[115] C. Li et al. "MCtandem: an efficient tool for large-scale peptide identification on many integrated core (MIC) architecture". In: *BMC Bioinformatics* 20.1 (Dec. 2019), p. 397. issn: 1471-2105. doi: 10.1186/s12859-019-2980-5 (cit. on p. 101).

[116] J. S. Liu and R. Chen. "Sequential monte carlo methods for dynamic systems". In: *Journal of the American Statistical Association* 93.443 (Sept. 1998), pp. 1032–1044. issn: 1537274X. doi: 10.1080/01621459.1998.10473765 (cit. on p. 28).

[117] F. Lopez et al. "Particle filtering on GPU architectures for manufacturing applications". In: *Computers in Industry* 71 (Aug. 2015), pp. 116–127. issn: 01663615. doi: 10.1016/j.compind.2015.03.013 (cit. on p. 30).

[118] J. F. Lucas. *Introduction to Abstract Mathematics*. Lanham, Maryland: Rowman & Littlefield, 1990. isbn: 978-0-912-67573-2 (cit. on p. 34).

[119] D. Luengo et al. "A survey of Monte Carlo methods for parameter estimation". In: *EURASIP Journal on Advances in Signal Processing* 2020.1 (Dec. 2020), p. 25. issn: 1687-6180. doi: 10.1186/s13634-020-00675-6 (cit. on p. 20).

[120] *Magma - Encyclopedia of Mathematics*. url: https://encyclopediaofmath.org/index.php?title=Magma (visited on 04/27/2022) (cit. on p. 35).

[121] P. Maheshwari and R. Albert. "Network model and analysis of the spread of Covid-19 with social distancing". In: *Applied Network Science* 5.1 (Dec. 2020), pp. 1–13. issn: 23648228. arXiv: 2006.09189. doi: 10.1007/s41109-020-00344-5 (cit. on p. 71).

[122] E. M. Marcotte. "How do shotgun proteomics algorithms identify proteins?" In: *Nature Biotechnology 2007 25:7* 25.7 (2007), pp. 755–757. issn: 1546-1696. doi: 10.1038/nbt0707-755 (cit. on p. 105).

[123] J. Marreiros, J. F. Gibaja Bao, and N. F. Bicho. *Use-wear and residue analysis in archaeology*. Springer, 2015. isbn: 9783319082578 (cit. on p. 57).

[124] N. L. Martisius et al. "Time wears on: Assessing how bone wears using 3D surface texture analysis". In: *PLOS ONE* 13.11 (Nov. 2018). Ed. by A. R. Evans, e0206078. issn: 1932-6203. doi: 10.1371/journal.pone.0206078 (cit. on p. 60).

[125] M. Matsumoto and T. Nishimura. “Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator”. In: *ACM Transactions on Modeling and Computer Simulation*. Vol. 8. 1. ACM PUB27 New York, NY, USA, Jan. 1998, pp. 3–30. doi: 10.1145/272991.272995 (cit. on p. 21).

[126] H. Matsuzoe. “Information geometry of Bayesian statistics”. In: *AIP Conference Proceedings*. Vol. 1641. 1. American Institute of Physics Inc., Feb. 2015, pp. 279–286. isbn: 9780735412804. doi: 10.1063/1.4905989 (cit. on p. 8).

[127] R. McElreath. *Statistical Rethinking - A Bayesian Course with Examples in R and STAN*. Boca Raton, Fla: CRC Press, 2020. isbn: 978-0-429-64231-9 (cit. on pp. 18, 63).

[128] F. Meier et al. “Online parallel accumulation–serial fragmentation (PASEF) with a novel trapped ion mobility mass spectrometer”. In: *Molecular and Cellular Proteomics* 17.12 (Dec. 2018), pp. 2534–2545. issn: 15359484. doi: 10.1074/mcp.TIR118.000900 (cit. on pp. 98, 113).

[129] D. Merkel. “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. In: *Linux J.* 2014.239 (2014). issn: 1075-3583 (cit. on p. 88).

[130] N. Metropolis and S. Ulam. “The Monte Carlo Method”. In: *Journal of the American Statistical Association* 44.247 (Sept. 1949), p. 335. issn: 01621459. doi: 10.2307/2280232 (cit. on p. 20).

[131] N. Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. doi: 10.1063/1.1699114 (cit. on p. 23).

[132] S. Minsker et al. *Scalable and Robust Bayesian Inference via the Median Posterior*. 2014. url: http://proceedings.mlr.press/v32/minsker14.html (visited on 02/03/2022) (cit. on p. 33).

[133] *Monoid - Encyclopedia of Mathematics*. url: https://encyclopediaofmath.org/wiki/Monoid (visited on 04/27/2022) (cit. on p. 34).

[134] D. C. Montgomery. *Design and Analysis of Experiments*. New York: Wiley, 2012. isbn: 978-1-118-09793-9 (cit. on p. 6).

[135] C. C. Moore. “Ergodic theorem, ergodic theory, and statistical mechanics”. In: *Proceedings of the National Academy of Sciences* 112.7 (Feb. 2015), pp. 1907–1911. issn: 0027-8424. doi: 10.1073/pnas.1421798112 (cit. on p. 22).

[136] S. Morita, P. F. Thall, and P. Müller. *Determining the effective sample size of a parametric prior*. June 2008. doi: 10.1111/j.1541-0420.2007.00888.x (cit. on pp. 9, 17).

[137] T. Müller-Gronbach, E. Novak, and K. Ritter. *Monte Carlo-Algorithmen*. Wiesbaden: Springer Berlin Heidelberg, 2012. isbn: 978-3-540-89140-6 (cit. on pp. 18, 20–23).

[138] A. Müller et al. “MetaCache: context-aware classification of metagenomic reads using minhashing”. In: *Bioinformatics* 33.23 (Dec. 2017). Ed. by I. Birol, pp. 3740–3748. issn: 1367-4803. doi: 10.1093/bioinformatics/btx520 (cit. on p. 101).

[139] W. J. Murdoch et al. “Definitions, methods, and applications in interpretable machine learning”. In: *Proceedings of the National Academy of Sciences* 116.44 (Oct. 2019), pp. 22071–22080. issn: 0027-8424. doi: 10.1073/pnas.1900654116 (cit. on p. 118).

[140] R. Neal. “MCMC Using Hamiltonian Dynamics”. In: *Handbook of Markov Chain Monte Carlo*. CRC Press, May 2011, pp. 113–162. isbn: 9781420079425. doi: 10.1201/b10905-6 (cit. on pp. 25, 26).

[141] R. Neal. “Sampling from multimodal distributions using tempered transitions”. In: *Statistics and Computing* 6.4 (1996), pp. 353–366. issn: 09603174. doi: 10.1007/BF00143556 (cit. on p. 34).

[142] W. Neiswanger, C. Wang, and E. Xing. “Asymptotically Exact, Embarrassingly Parallel MCMC”. In: *Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014* (Nov. 2013), pp. 623–632. url: http://arxiv.org/abs/1311.4780 (cit. on p. 33).

[143] J. A. Nelder and R. W. M. Wedderburn. “Generalized Linear Models”. In: *Journal of the Royal Statistical Society. Series A (General)* 135.3 (1972), p. 370. issn: 00359238. doi: 10.2307/2344614 (cit. on p. 60).

[144] C. Nemeth and C. Sherlock. “Merging MCMC Subposteriors through Gaussian-Process Approximations”. In: *Bayesian Analysis* 13.2 (June 2018), pp. 507–530. issn: 1936-0975. doi: 10.1214/17-BA1063 (cit. on p. 33).

[145] B. Neuenschwander et al. “Predictively Consistent Prior Effective Sample Sizes”. In: *Biometrics* (July 2019). url: http://arxiv.org/abs/1907.04185 (cit. on p. 9).

[146] J. von Neumann. “Various Techniques Used in Connection with Random Digits”. In: *Monte Carlo Method*. Ed. by A. Householder, G. Forsythe, and H. Germond. Vol. 12. National Bureau of Standards Applied Mathematics Series. Washington, DC: US Government Printing Office, 1951. Chap. 13, pp. 36–38 (cit. on p. 21).

[147] V. K. Nguyen et al. “Analysis of Practical Identifiability of a Viral Infection Model”. In: *PLOS ONE* 11.12 (Dec. 2016). Ed. by J. Vera, e0167568. issn: 1932-6203. doi: 10.1371/journal.pone.0167568 (cit. on p. 73).

[148] A. Nicoll et al. “Experience and lessons from surveillance and studies of the 2009 pandemic in Europe”. In: *Public Health* 124.1 (Jan. 2010), pp. 14–23. issn: 00333506. doi: 10.1016/j.puhe.2009.12.001 (cit. on p. 71).

[149] J. A. Novella et al. “Container-based bioinformatics with Pachyderm”. In: *Bioinformatics* 35.5 (2018), pp. 839–846. issn: 1367-4803. doi: 10.1093/bioinformatics/bty699 (cit. on pp. 77, 83, 88).

[150] B. Nunes, I. Natário, and M. Lucília Carvalho. “Nowcasting influenza epidemics using non-homogeneous hidden Markov models”. In: *Statistics in Medicine* 32.15 (July 2013), pp. 2643–2660. issn: 02776715. doi: 10.1002/sim.5670 (cit. on p. 73).

[151] G. Nuti, L. A. J. Rugama, and A.-I. Cross. “A Bayesian Decision Tree Algorithm”. Jan. 2019. url: http://arxiv.org/abs/1901.03214 (cit. on p. 92).

[152] L. Page and S. Brin. “The anatomy of a large-scale hypertextual Web search engine”. In: *Computer Networks* 30.1-7 (Jan. 1998), pp. 107–117. issn: 13891286. doi: 10.1016/s0169-7552(98)00110-x (cit. on p. 21).

[153] Pallets. *Flask*. URL: https://flask.palletsprojects.com/en/2.0.x/ (cit. on pp. 83, 87).

[154] F. Panneton, P. L'Ecuyer, and M. Matsumoto. "Improved long-period generators based on linear recurrences modulo 2". In: *ACM Transactions on Mathematical Software* 32.1 (Mar. 2006), pp. 1–16. ISSN: 00983500. DOI: 10.1145/1132973.1132974 (cit. on p. 21).

[155] S. Y. Park and A. K. Bera. "Maximum entropy autoregressive conditional heteroskedasticity model". In: *Journal of Econometrics* 150.2 (June 2009), pp. 219–230. ISSN: 03044076. DOI: 10.1016/j.jeconom.2008.12.014 (cit. on p. 63).

[156] P. Pawlus, R. Reizer, and M. Wieczorowski. "Functional Importance of Surface Texture Parameters". In: *Materials* 14.18 (Sept. 2021), p. 5326. ISSN: 1996-1944. DOI: 10.3390/ma14185326 (cit. on p. 59).

[157] A. Pedergnana et al. "Evaluating the microscopic effect of brushing stone tools as a cleaning procedure". In: *Quaternary International* (July 2020). ISSN: 10406182. DOI: 10.1016/j.quaint.2020.06.031 (cit. on pp. 28, 57, 58, 64, 65, 67).

[158] A. Pedergnana et al. "Polish is quantitatively different on quartzite flakes used on different worked materials". In: *PLOS ONE* 15.12 (Dec. 2020). Ed. by M. Peresani, e0243295. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0243295 (cit. on pp. 58, 115–117, 119, 124).

[159] F. Pedregosa et al. "Scikit-Learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12 (Nov. 2011), pp. 2825–2830. ISSN: 1532-4435 (cit. on pp. 117, 118, 122).

[160] O. Peters and M. Gell-Mann. "Evaluating gambles using dynamics". In: *Chaos* 26.2 (Feb. 2016), p. 023103. ISSN: 10541500. DOI: 10.1063/1.4940236 (cit. on p. 22).

[161] L. K. Pino et al. *Emerging mass spectrometry-based proteomics methodologies for novel biomedical applications*. Oct. 2020. DOI: 10.1042/BST20191091 (cit. on p. 97).

[162] Plotly. *Dash*. URL: https://dash.plotly.com/ (cit. on pp. 83, 87).

[163] F. P. Polack et al. "Safety and Efficacy of the BNT162b2 mRNA Covid-19 Vaccine". In: *New England Journal of Medicine* 383.27 (Dec. 2020), pp. 2603–2615. ISSN: 0028-4793. DOI: 10.1056/nejmoa2034577 (cit. on p. 15).

[164] K. R. Popper. *Logik der Forschung*. Tübingen: Mohr Siebeck, 2005. ISBN: 978-3-161-48111-6 (cit. on p. 6).

[165] M. L. de Prado. *Advances in Financial Machine Learning*. New York: John Wiley & Sons, 2018. ISBN: 978-1-119-48208-6 (cit. on p. 6).

[166] M. M. L. de Prado. *Machine Learning for Asset Managers*. Cambridge: Cambridge University Press, 2020. ISBN: 978-1-108-88540-9 (cit. on p. 116).

[167] W. H. Press et al. *Numerical Recipes - The Art of Scientific Computing*. 3rd Edition. Cambridge: Cambridge University Press, 2007. ISBN: 978-0-521-88068-8 (cit. on pp. 18, 21).

[168] N. Prianichnikov et al. "Maxquant software for ion mobility enhanced shotgun proteomics". In: *Molecular and Cellular Proteomics* 19.6 (Mar. 2020), pp. 1058–1069. ISSN: 15359484. DOI: 10.1074/mcp.TIR119.001720 (cit. on p. 101).

[169] J. R. Quinlan. “Induction of decision trees”. In: *Machine Learning* 1.1 (Mar. 1986), pp. 81–106. issn: 0885-6125. doi: 10.1007/bf00116251 (cit. on p. 118).

[170] M. Quiroz et al. “Speeding Up MCMC by Efficient Data Subsampling”. In: *Journal of the American Statistical Association* 114.526 (Apr. 2019), pp. 831–843. issn: 1537274X. doi: 10.1080/01621459.2018.1448827 (cit. on p. 33).

[171] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Cambridge: MIT Press, 2005. isbn: 978-0-262-18253-9 (cit. on pp. 36, 48).

[172] J. Revels. *BenchmarkTools.jl, Version 1.25*. 2021. url: https://juliaci.github.io/BenchmarkTools.jl/dev/ (cit. on p. 51).

[173] B. C. Ross. “Mutual information between discrete and continuous data sets”. In: *PLoS ONE* 9.2 (Feb. 2014), e87357. issn: 19326203. doi: 10.1371/journal.pone.0087357 (cit. on p. 117).

[174] M. Salmon, D. Schumacher, and M. Höhle. “Monitoring count time series in R: Aberration detection in public health surveillance”. In: *Journal of Statistical Software* 70 (May 2016), pp. 1–35. issn: 15487660. arXiv: 1411.1292. doi: 10.18637/jss.v070.i10 (cit. on p. 77).

[175] M. Salmon et al. “A system for automated outbreak detection of communicable diseases in Germany”. In: *Eurosurveillance* 21.13 (Mar. 2016), p. 30180. issn: 1560-7917. doi: 10.2807/1560-7917.ES.2016.21.13.30180 (cit. on p. 77).

[176] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck. “Probabilistic programming in Python using PyMC3”. In: *PeerJ Computer Science* 2 (Apr. 2016), e55. issn: 2376-5992. doi: 10.7717/peerj-cs.55 (cit. on pp. 28, 41–43, 46).

[177] B. Sartorius, A. B. Lawson, and R. L. Pullan. “Modelling and predicting the spatio-temporal spread of COVID-19, associated deaths and impact of key risk factors in England”. In: *Scientific Reports* 11.1 (Dec. 2021), p. 5378. issn: 2045-2322. doi: 10.1038/s41598-021-83780-2 (cit. on p. 72).

[178] M. J. Schervish. *Theory of Statistics*. Springer Series in Statistics. New York, NY: Springer New York, 1995. isbn: 978-1-4612-8708-7. doi: 10.1007/978-1-4612-4250-5 (cit. on p. 8).

[179] R. Schlickeiser and M. Kröger. “Analytical solution of the SIR-model for the temporal evolution of epidemics: Part B. Semi-time case”. In: *Journal of Physics A: Mathematical and Theoretical* 54.17 (Apr. 2021), p. 175601. issn: 17518121. doi: 10.1088/1751-8121/abed66 (cit. on p. 70).

[180] E. Schrödinger. “An undulatory theory of the mechanics of atoms and molecules”. In: *Physical Review* 28.6 (Dec. 1926), pp. 1049–1070. issn: 0031899X. doi: 10.1103/PhysRev.28.1049 (cit. on p. 6).

[181] S. L. Scott et al. “Bayes and Big Data: The Consensus Monte Carlo Algorithm”. In: *International Journal of Management Science and Engineering Management* 11 (2016), pp. 78–88. url: http://www.tandfonline.com/doi/full/10.1080/17509653.2016.1142191 (cit. on p. 33).

[182] H.-J. Seelos, ed. *Medizinische Informatik, Biometrie und Epidemiologie*. DE GRUYTER, Dec. 1997. isbn: 978-3-11-014317-1. doi: 10.1515/9783110809329 (cit. on p. 69).

[183] M. W. Senko, S. C. Beu, and F. W. McLafferty. "Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions". In: *Journal of the American Society for Mass Spectrometry* 6.4 (Apr. 1995), pp. 229–233. issn: 1044-0305. doi: 10.1016/1044-0305(95)00017-8 (cit. on pp. 98, 112, 114).

[184] C. E. Shannon. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423. issn: 15387305. doi: 10.1002/j.1538-7305.1948.tb01338.x (cit. on pp. 63, 117).

[185] M. Slawski et al. "Isotope pattern deconvolution for peptide mass spectrometry by non-negative least squares/least absolute deviation template matching". In: *BMC Bioinformatics* 13.1 (Dec. 2012), p. 291. issn: 1471-2105. doi: 10.1186/1471-2105-13-291 (cit. on p. 101).

[186] A. J. Smola and B. Schölkopf. "A tutorial on support vector regression". In: (2004). url: http://citeseerx.ist.psu.edu/viewdoc/summary;jsessionid=CFFC9E2D2290417A7DC1A46B16DC3F6C?doi=10.1.1.114.4288 (cit. on p. 120).

[187] H. Snoussi. "Bayesian Information Geometry: Application to Prior Selection on Statistical Manifolds". In: *Advances in Imaging and Electron Physics*. Vol. 146. Academic Press Inc., Jan. 2007, pp. 163–207. doi: 10.1016/S1076-5670(06)46003-1 (cit. on p. 63).

[188] M. D. Stasinopoulos et al. *Flexible Regression and Smoothing*. Chapman and Hall/CRC, Apr. 2017, pp. 1–549. isbn: 9781315269870. doi: 10.1201/b21973 (cit. on p. 72).

[189] O. Stojanović et al. "A Bayesian Monte Carlo approach for predicting the spread of infectious diseases". In: *PLOS ONE* 14.12 (Dec. 2019). Ed. by C. W. Chen, e0225838. issn: 1932-6203. doi: 10.1371/journal.pone.0225838 (cit. on pp. 73, 77).

[190] Y. Sugita and Y. Okamoto. "Replica-exchange molecular dynamics method for protein folding". In: *Chemical Physics Letters* 314.1-2 (Nov. 1999), pp. 141–151. issn: 00092614. doi: 10.1016/S0009-2614(99)01123-9 (cit. on p. 34).

[191] R. H. Swendsen and J. S. Wang. "Replica Monte Carlo simulation of spin-glasses". In: *Physical Review Letters* 57.21 (Nov. 1986), pp. 2607–2609. issn: 00319007. doi: 10.1103/PhysRevLett.57.2607 (cit. on p. 34).

[192] N. N. Taleb. *Incerto: Fooled by Randomness, The Black Swan, The Bed of Procrustes, Antifragile*. Random House Publishing Group, 2016. isbn: 9780399590450 (cit. on p. 6).

[193] N. N. Taleb. "Statistical Consequences of Fat Tails: Real World Preasymptotics, Epistemology, and Applications". Jan. 2020. url: http://arxiv.org/abs/2001.10488 (cit. on p. 63).

[194] The Cloud Native Computing Foundation. *Helm*. url: https://www.helm.sh/ (cit. on p. 89).

[195] The Kubernetes Authors. *Kubernetes*. url: https://kubernetes.io (cit. on p. 83).

[196] The Theano Development Team et al. "Theano: A Python framework for fast computation of mathematical expressions". May 2016. url: http://arxiv.org/abs/1605.02688 (cit. on p. 42).

[197] T. R. Thomas et al. “Traceology, quantifying finishing machining and function: A tool and wear mark characterisation study”. In: *Wear* 271.3-4 (June 2011), pp. 553–558. issn: 00431648. doi: 10.1016/j.wear.2010.04.025 (cit. on p. 57).

[198] *Unital – from Wolfram MathWorld*. url: https://mathworld.wolfram.com/Unital.html (visited on 04/27/2022) (cit. on p. 35).

[199] D. Valkenborg et al. “The isotopic distribution conundrum”. In: *Mass Spectrometry Reviews* 31.1 (Jan. 2012), pp. 96–109. issn: 02777037. doi: 10.1002/mas.20339 (cit. on p. 98).

[200] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. isbn: 1441412697 (cit. on p. 41).

[201] G. Van Rossum and F. L. Drake. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995 (cit. on p. 41).

[202] C. L. Vidal Rodeiro and A. B. Lawson. “Online updating of space-time disease surveillance models via particle filters”. In: *Statistical Methods in Medical Research* 15.5 (Oct. 2006), pp. 423–444. issn: 09622802. doi: 10.1177/0962280206071640 (cit. on pp. 50, 73).

[203] L. Wang, S. Li, and H. Tang. “msCRUSH: Fast Tandem Mass Spectral Clustering Using Locality Sensitive Hashing”. In: *Journal of Proteome Research* (Dec. 2018), acs.jproteome.8b00448. issn: 1535-3893. doi: 10.1021/acs.jproteome.8b00448 (cit. on pp. 101, 105).

[204] L. Wang et al. “A Fast and Memory-Efficient Spectral Library Search Algorithm Using Locality-Sensitive Hashing”. In: *Proteomics* 20.21-22 (Nov. 2020). issn: 16159861. doi: 10.1002/pmic.202000002 (cit. on pp. 101, 105).

[205] X. Wang and D. B. Dunson. “Parallelizing MCMC via Weierstrass Sampler”. Dec. 2013. url: http://arxiv.org/abs/1312.4605 (cit. on p. 33).

[206] A. D. Weston and L. Hood. “Systems Biology, Proteomics, and the Future of Health Care: Toward Predictive, Preventative, and Personalized Medicine”. In: *Journal of Proteome Research* 3.2 (Apr. 2004), pp. 179–196. issn: 1535-3893. doi: 10.1021/pr0499693 (cit. on p. 97).

[207] M. Wiesenfarth and S. Calderazzo. “Quantification of prior impact in terms of effective current sample size”. In: *Biometrics* 76.1 (Mar. 2020), pp. 326–336. issn: 0006-341X. doi: 10.1111/biom.13124 (cit. on p. 9).

[208] E. P. Wigner. “The unreasonable effectiveness of mathematics in the natural sciences. Richard courant lecture in mathematical sciences delivered at New York University, May 11, 1959”. In: *Communications on Pure and Applied Mathematics* 13.1 (Feb. 1960), pp. 1–14. issn: 00103640. doi: 10.1002/cpa.3160130102 (cit. on p. 5).

[209] J. R. Wiśniewski et al. “Universal sample preparation method for proteome analysis”. In: *Nature Methods* 6.5 (2009), pp. 359–362. issn: 15487091. doi: 10.1038/nmeth.1322 (cit. on p. 113).

[210] Z. Xia et al. “A Privacy-Preserving and Copy-Deterrence Content-Based Image Retrieval Scheme in Cloud Computing”. In: *IEEE Transactions on Information Forensics and Security* 11.11 (Nov. 2016), pp. 2594–2608. issn: 1556-6013. doi: 10.1109/TIFS.2016.2590944 (cit. on p. 101).

[211] M. Zaharia et al. “Spark : Cluster Computing with Working Sets”. In: *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (2010) (cit. on pp. 35, 43, 109).

[212] Y. Zhang et al. “Video anomaly detection based on locality sensitive hashing filters”. In: *Pattern Recognition* 59 (Nov. 2016), pp. 302–311. issn: 00313203. doi: 10.1016/j.patcog.2015.11.018 (cit. on p. 101).

[213] Y. Zhou. “vSMC: Parallel Sequential Monte Carlo in C++”. In: *Journal of Statistical Software* 62.9 (June 2013), pp. 1–49. url: http://arxiv.org/abs/1306.5583 (cit. on p. 30).

[214] F. T. Zohora et al. “DeepIso: A Deep Learning Model for Peptide Feature Detection from LC-MS map”. In: *Scientific Reports* 9.1 (Dec. 2019), pp. 1–13. issn: 20452322. doi: 10.1038/s41598-019-52954-4 (cit. on p. 101).

[215] J. Zou, Z. Zhang, and H. Yan. “A hybrid hierarchical Bayesian model for spatiotemporal surveillance data”. In: *Statistics in Medicine* 37.28 (Dec. 2018), pp. 4216–4233. issn: 02776715. doi: 10.1002/sim.7909 (cit. on p. 73).

Konstantin Bob

Lebenslauf

[Removed in online version]