Novel Applications of Machine Learning and Data Science Methods in Evolutionary Genomics

Zur Erlangung des akademischen Grades einer

Doktorin der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Julia Annette Maria Haag

Tag der mündlichen Prüfung: 29.07.2025

1. Referent/Referentin: Prof. Dr. Alexandros Stamatakis
2. Referent/Referentin: Prof. Dr. Arndt von Haeseler

Hiermit erkläre ich, dass ich diese Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe. Ich habe die Satzung des Karlsruher Institutes für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis beachtet.

Karlsruhe, 29.07.2025

. . . . . .

(**Julia Haag**)

Abstract

The field of evolutionary genomics studies the evolutionary mechanisms that lead to the broad diversity of life on earth. Understanding these mechanisms not only helps to disentangle the origin of life, but also has practical applications such as for developing novel drugs, or tracking global pandemics. Phylogenetics and population genetics are two major subfields of evolutionary biology. While phylogenetics focuses on the evolutionary history between distinct species, population genetics studies the evolutionary mechanisms within populations of a single species, or among closely related species. In modern-day evolutionary genomics, analyses are typically performed using molecular sequence data and rely upon mathematical models that are implemented in scientific software tools.

Over the past decades, technological improvements have led to an avalanche of molecular sequence data. The amount of available data increases at a higher pace than the cost of compute power decreases, according to Moore's law. Consequently, data need to be selected systematically for analysis. In addition, analysis methods need to be as fast as possible while maintaining analytical accuracy. Furthermore, quantifying our confidence in these analytical findings becomes increasingly important to prevent misleading conclusions based on potentially uncertain results.

In this thesis, I explore applications of machine learning and data science methods to contribute towards conducting more systematic data selection and faster data analyses, as well as towards better quantifying the inherent uncertainty of such analyses.

My first contribution is a machine learning-based framework that predicts the difficulty of phylogenetic analyses. Usually, phylogenetic trees that represent the hypothetical evolutionary history among a set of distinct species are inferred via complex mathematical models such as the Maximum Likelihood (ML) criterion. These inference procedures are time- and resource-intensive and rely on heuristic tree inference algorithm implementations. Performing multiple independent inferences is often required to at least partially explore the vast search space of possible tree topologies. Yet, these independent inferences do not necessarily converge to a single phylogeny or even topologically highly similar trees. I initially present a novel quantification of this behavior and validate that this quantification accurately represents the inherent dataset difficulty under ML phylogenetic analyses. I further present a machine learning model that can predict this difficulty with high accuracy while being substantially faster than even a single ML tree inference.

My second contribution is a collaborative study of sequence simulation realism conducted with Johanna Trost and Dimitri Höhler. In an extensive data analysis, we assessed the degree of realism for sequence simulation in phylogenetics. These data simulations rely on statistical models of evolution. A plethora of distinct models exists, comprising simple to highly complex models with numerous degrees of freedom. We simulate data under increasingly complex evolutionary models via state-of-the art sequence simulation software. Using two distinct machine learning-based classification methods, we demonstrate that, across all models, we can easily distinguish empirical from simulated data with high accuracy. This indicates a lack of realism in sequence simulation. We further conduct thorough analyses to explain why current state-of-the art simulations fail to replicate empirical data.

My final contribution is a software tool for uncertainty estimation in dimensionality reduction for population genetics data. High-dimensional genetic data often require the application of dimensionality reduction techniques to identify patterns and facilitate their interpretation. While there exist numerous software tools for performing dimensionality reduction, none of these tools deploy uncertainty estimation. With *Pandora*, I present a novel framework that seamlessly integrates dimensionality reduction of population genetics data and the respective uncertainty estimation into a single software tool. Pandora estimates the uncertainty based on a bootstrapping approach and calculates three distinct notions of stability. Through comprehensive evaluations on empirical and simulated data, I demonstrate the usage and utility of Pandora for studies that rely on dimensionality reduction techniques.

Acknowledgments

First, I would like to thank my primary supervisor, Prof. Dr. Alexandros Stamatakis. Thank you for the guidance and unconditional support over the past years. The close supervision, not only in my PhD but also during my Master's thesis, always felt like a privilege, and academia would be a better place if more professors had such a level of dedication to their students. I would also like to thank Prof. Dr. Arndt von Haeseler who kindly agreed to co-supervise my thesis.

Furthermore, I wish to thank my colleagues from the Exelixis Lab for the years of fruitful discussions and fun coffee chats: Alexey Kozlov, Benoit Morel, Ben Bettisworth, Dimitri Höhler, Lukas Hübner, Anastasis Togkousidis, Luise Häuser, and Johannes Hengstler. I'm also grateful to my collaborators: Johanna Trost, Bastien Boussau, Laurent Jacob, and Alexander I. Jordan, for their invaluable contributions.

With all my heart, I thank my parents, Annette and Karl-Heinz Schmid, whose unwavering love and support have been the foundation of my journey. Your encouragement and belief in me have been invaluable, and I am truly blessed to have such wonderful parents.

I want to express my sincerest gratitude to my husband, Jonas. You've been by my side for the past 10 years, always there to lift me up when I was down and to celebrate every little win. I couldn't have done this without your love and support, and I'm excited to continue this life's journey with you.

I want to thank all my friends who kept me grounded and sane throughout my years of study and my PhD. Your endless hours of fun, chaos, and laughter made all the difference. A special shout-out to Paul, Lorenz, Max, Julien, Björn, Marcel, Malte, Rafael, Dave, Fabi, Mark, Daniel, Lisa, Philipp, Flo, Ella, Bela, Jan, and Alex. I'm so grateful to have shared this experience with you all.

Finally, I would like to thank the Klaus Tschira Foundation for funding me and my research. I thank the Heidelberg Institute for Theoretical Studies and its entire staff for providing such an excellent working place.

Contents

List of Figures

List of Tables

List of Acronyms

AA Amino Acid.

ACC Accuracy.

BACC Balanced Accuracy.

CNN Convolutional Neural Network.

DNA Deoxyribonucleic Acid.

GBT Gradient Boosted Trees.

GTR General Time Reversible.

Indel Insertion and Deletion.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MDS Multidimensional Scaling.

ML Maximum Likelihood.

MP Maximum Parsimony.

MSA Multiple Sequence Alignment.

MSE Mean Squared Error.

NNI Nearest Neighbor Interchange.

PCA Principal Component Analysis.

RF-Distance Robinson-Foulds Distance.

RNA Ribonucleic Acid.

SNP Single Nucleotide Polymorphism.

SPR Subtree Pruning and Regrafting.

TBR Tree Bisection and Reconnection.

1. Introduction

1.1 Background and Motivation

All life on earth is fundamentally composed of only four molecules: adenine, cytosine, guanine, and thymine. Linking billions of these molecules, the Deoxyribonucleic Acid (DNA) encodes the instructions for the development, functioning, and reproduction of all living organisms. This genetic "code of life" is the foundation for the incredible diversity of life we observe, from bacteria to complex life forms such as ourselves.

A key role in shaping this diversity is *evolution*. Evolution is the change of heritable characteristics over time as a result of mutation, genetic drift, gene flow, and natural selection. Random mutations change the DNA, and random events can change the genetic composition within a population. Organisms migrate between populations, inducing an exchange of genetic material (gene flow). Finally, natural selection favors organisms with traits that are more advantageous in a certain environment. Through these processes, life on earth has evolved from simple single-celled organisms to the vast array of complex life forms we observe today.

Evolutionary genomics strives to understand the evolutionary processes leading to this broad diversity. Studying evolution is not only relevant for disentangling the origin of life [189], but can also help to track pandemics [76], or solve criminal cases [119]. *Phylogenetics* and *population genetics* are two major subfields of evolutionary biology. Phylogenetics studies the evolutionary history between a set of distinct species. Population genetics focuses on the genetic composition and its change over time between individuals of a single species or among closely related species.

With the advent of modern DNA sequencing technology, researchers can obtain the entire genetic material (*genome*) of an organism. Over the past decades, sequencing techniques have become increasingly faster and cheaper, leading to an avalanche of data that need to be analyzed. For instance, the number of sequences in GenBank, a widely used public sequence database, still increases exponentially [127], and the cost

of sequencing a human genome decreases faster than the computing cost according to Moore's law [128]. This discrepancy between available genomic data and compute power does not only require a systematic analysis of these data, but also calls for faster analysis methods.

With these vast amounts of data and numerous approaches to analyze these data, it becomes increasingly important to quantify the confidence in findings to prevent potentially misleading conclusions based on uncertain results. However, in modern-day science, these data are analyzed using highly complex mathematical models, which are typically implemented in scientific software tools. Quantifying the uncertainty of findings, and integrating uncertainty estimations into software, constitutes a crucial aspect of method development and software engineering. Yet, many analysis methods and software tools still lack the capability to estimate uncertainty.

Methods from machine learning and data science offer promising solutions to these problems. The systematic statistical analysis of data, commonly known as *data science*, allows for extracting meaningful insights, for instance, regarding uncertainty of results. Machine learning models can process large amounts of data and can detect previously unknown patterns. Across fields, researchers have successfully deployed machine learning-based solutions to previously unsolved tasks. One of the most-widely known examples in Bioinformatics is *AlphaFold* [86]. Entering the Critical Assessment of Protein Structure Prediction (CASP) competition for 3D protein structure prediction in 2020, AlphaFold outperformed all competing computational methods by a large margin, and demonstrated prediction accuracy comparable to experimentally determined structures [97].

The goal of this thesis is to explore potential applications of data science and machine learning to problems in phylogenetics and population genetics. My focus lies on systematic and fast phylogenetic analyses, as well as on uncertainty estimation for population genetics.

1.2 Scientific Contribution

My main contributions in the context of this thesis are the open-source tools *Pythia*, *Pandora*, and a thorough study of the realism of data simulations in phylogenetics.

Pythia is a machine learning-based framework that predicts the degree of *difficulty* of phylogenetic analyses. The input for a phylogenetic analysis is a set of biological sequences (for instance DNA sequences) representing multiple organisms, and the result is a *phylogenetic tree* that represents their inferred hypothetical evolutionary history. The input sequences are assembled into a data structure called *Multiple Sequence Alignment* (MSA). For reasons, I will further detail in Chapter 2, inferring a phylogenetic tree based on an MSA requires heuristic search algorithms. Consequently, we typically infer multiple phylogenetic trees on the same MSA to obtain a broad sampling of the vast space of possible tree topologies. On *easy* MSAs, multiple tree inferences converge to highly similar tree topologies, indicating a clear signal towards a certain evolutionary history. In contrast, *difficult* MSAs yield highly distinct, yet equally well-supported topologies. Accepting a single evolutionary history

of species based on a difficult MSA is thus strongly discouraged. Inferring multiple phylogenetic trees, and performing post-analyses to determine the degree of difficulty of an MSA, is time- and resource-intensive. In my work, I developed a quantification of this behavior and demonstrated that this quantification accurately represents the degree of phylogenetic inference difficulty for a given MSA. I additionally developed Pythia, a machine learning model that predicts the difficulty based on fast-to-compute features, *before* conducting any time-consuming phylogenetic analyses. My first version of Pythia was published in the journal of *Molecular Biology and Evolution* [66]. Since this initial publication in 2022, I introduced various changes to improve the prediction performance as well as the runtime of Pythia. I recently published these changes on the *bioRxiv* preprint server [65].

With *Pandora*, I developed a framework to estimate uncertainty of dimensionality reduction performed on population genetics data. Population genetics data is high-dimensional, with data typically comprising thousands of dimensions. To identify patterns of similarity between individuals in these data, and to allow for visualization and interpretation, dimensionality reduction techniques project the data into a lower-dimensional space. While a plethora of tools exists for conducting this dimensionality reduction, there does not exist a tool for population genetics data that can provide an uncertainty estimate. Yet, such an uncertainty estimate is important to prevent overconfident conclusions in downstream analyses that are based on dimensionality reduction results. My novel Pandora tool provides uncertainty estimates for two frequently applied dimensionality reduction approaches based on bootstrapping. I performed thorough analyses using empirical and simulated population genetics data to demonstrate the usability and utility of Pandora in detecting instability in dimensionality reduction. Pandora was published in the journal of *Bioinformatics Advances* [68].

MSA simulation is an important tool in phylogenetics if a ground-truth phylogenetic tree is required. MSA simulations rely on statistical models of evolution that aim to model the process of biological sequence evolution (see Chapter 2). Ideally, simulated MSAs mimic empirical MSAs as closely as possible. In a thorough study, Johanna Trost, Dimitri Höhler, and I analyzed the degree of realism of sequence simulations under a plethora of evolutionary models. We deployed two distinct types of machine learning classification models to distinguish between simulated and empirical MSAs. We demonstrated high prediction accuracy across all evolutionary models, indicating low simulation realism. Additionally, we provided thorough analyses to explain why current state-of-the art simulations fail to generate empirical-like MSAs. Our work was published in the journal of *Molecular Biology and Evolution* [178].

Maximum Likelihood (ML) is a popular phylogenetic tree inference method. Inferring trees under ML requires heuristic search algorithms (see Chapter 2). These heuristics internally rely on numerical optimization routines, and their convergence is determined based on several numerical thresholds. I systematically analyzed the impact of these thresholds on the log-likelihood and runtime of tree inferences in three popular ML inference tools. I used two collections of MSAs (*Data collection 1* and *Data collection 2*) to quantify the runtime and quality impact. For IQ-

TREE [121], I identified a threshold setting optimization that yields an inference time speedup without impacting the quality of the inferred phylogenetic trees. I observed a speedup of $1.3 \pm 0.4\times$ on *Data collection 1* and $1.3 \pm 0.9\times$ on *Data collection 2*. For RAxML-NG [96], I identified two such improved threshold settings with a speedup of $1.9 \pm 0.6\times$ on *Data collection 1* and $1.8 \pm 1.1\times$ on *Data collection 2*. Using *Data collection 2*, I further analyzed the impact of numerical thresholds on the results and runtime of the RAxML-NG bootstrap procedure. With this bootstrap procedure, RAxML-NG estimates the statistical confidence for the inner branches of a given phylogenetic tree. Adjusting the settings for two numerical thresholds in RAxML-NG results in an average speedup of $1.9 \pm 0.8\times$ without influencing the resulting confidence estimates. The results of this study are published in the journal of *Bioinformatics Advances* [67]. Since I conducted parts of these analyses during my Master's thesis, I refrained from including this project in this thesis to provide a clear separation between both theses.

Over the course of my research for this thesis, I also contributed to several projects that resulted in peer-reviewed publications, but have not been included here.

Based on my difficulty predictor Pythia, Togkousidis *et al.* [175] developed a difficulty-aware version of the ML tree inference tool RAxML-NG. Depending on the predicted difficulty of the given input MSA, *adaptive RAxML-NG* executes a different tree inference heuristic. These changes result in a substantial speedup while not affecting the quality of the inferred tree.

Furthermore, I contributed to the development of the *Educated Bootstrap Guesser* (EBG) [188], a machine learning-based framework that predicts bootstrap support values on phylogenetic trees. Bootstrap support values measure the confidence of individual inner branches in a phylogenetic tree. Typically, these support values are computed using the *Felsenstein Bootstrap* approach [48], which relies on time- and resource-intensive phylogenetic tree inferences. Instead, EBG predicts the support based on fast-to-compute features and provides an accurate approximation of the inner branch confidence values in phylogenetic trees.

Finally, I participated in a study on the evolutionary history of avian lineages by analyzing the difficulty of vast amounts of MSAs using my Pythia prediction tool. This study is published in *Nature* [170].

1.3 Structure and Overview

This thesis is structured as follows. In Chapter 2, I introduce the fundamental concepts of phylogenetics, population genetics, and machine learning that are relevant to this work. In Chapter 3, I describe Pythia, a machine learning-based difficulty prediction framework for phylogenetics. In Chapter 4, I present the study on the realism of phylogenetic sequence simulations, which I conducted in collaboration with Johanna Trost and Dimitri Höhler. In Chapter 5, I describe Pandora, a tool to estimate dimensionality reduction stability of genotype data. Finally, I conclude and discuss future work in Chapter 6.

2. Fundamentals

2.1 Phylogenetics

Evolution is the gradual change of heritable characteristics from generation to generation. The driving forces of evolution are mutation, natural selection, gene flow, and genetic drift. Evolution results in changes at the molecular and, as a consequence, at the morphological level as well. In a *speciation* event, one species splits into two new species that continue to evolve independently, whereas in an *extinction event*, a species goes extinct. The field of *Phylogenetics* studies this evolutionary history among a set of organisms or species, and the history is typically visualized as a tree structure called *phylogenetic tree* or *phylogeny* for short. The inner nodes of this tree represent putative speciation events and correspond to hypothetical common ancestors. The leaves represent the (extant) organisms under study and are called *taxa*. Branch lengths within a phylogenetic tree typically indicate the relative evolutionary distance between two nodes. Figure 2.1 shows an example of a phylogenetic tree of birds.

Phylogenetic trees are usually strictly binary, meaning that a speciation event always splits a lineage into two sublineages. A lineage is a continuous line of descent tracing the evolutionary path from a common ancestor to its descendants. Multifurcation events, that is, a lineage splitting into more than two sublineages can be represented via near-zero branch lengths.

Phylogenetic trees can be rooted or unrooted. In rooted phylogenetic trees, the root node represents the most recent common ancestor of all organisms contained in the phylogenetic tree. In contrast, unrooted phylogenetic trees only show the hypothetical speciation history without identifying a common ancestor. In this thesis, we focus on unrooted, binary trees.

Phylogenetic inference methods have been mainly developed for biological sequence data. They are, however, also popular outside the context of biology, for example in

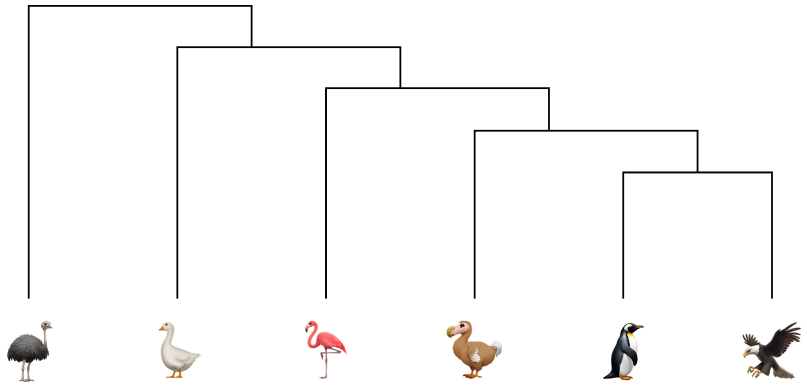


Figure 2.1: Phylogenetic tree of birds based on Stiller *et al.* [170].

linguistics to infer the evolutionary history of natural languages [11]. In this thesis, we focus on biological sequence data.

2.1.1 Biological Sequence Data

Phylogenetic trees are inferred using either *phenotypic* or *genotypic* data, or a combination of both.

Phenotypic data consists of observable characteristics or traits of the organisms under study. Typically, morphological traits are used for tree inference, for example, the shape and size of leaves in plants, or the presence or absence of specific physical features in animals. The phylogeny is then inferred based on the similarities of these traits. Morphological traits can be obtained via manual extraction by experts. More recent approaches using three-dimensional imaging techniques, such as micro-computed tomography, can automate trait extraction and increase the number of traits available for phylogenetic inference [45]. The resulting data is a matrix, where each row represents one taxon and each column represents one trait. Traits can be encoded as being present or absent (typically indicated as 1 and 0 respectively). Alternatively, a trait can assume multiple values, if the trait captures, for example, the color of a flower.

Genotypic data consist of genetic information, typically either Deoxyribonucleic Acid (DNA), Ribonucleic Acid (RNA), or Amino Acid (AA) sequences, or a combination thereof. These sequences can be automatically extracted from a tissue sample using next-generation sequencing (NGS) techniques [82]. The resulting data are sequences of DNA, RNA, or AA characters. For DNA and RNA, these characters correspond to the four nucleotides (`A`, `C`, `G`, and `T`/`U`), and 10 additional characters for partially ambiguous data (`W`, `S`, `M`, `K`, `R`, `Y`, `B`, `D`, `H`, `V`). For example, the character `W` translates to "either `A` or `T`". For AA data the characters encode the 20 amino acids (`A`, `R`, `N`, `D`, `C`, `Q`, `E`, `G`, `H`, `I`, `L`, `K`, `M`, `F`, `P`, `S`, `T`, `W`, `Y`, `V`), with two additional characters for partially ambiguous AAs (`B`, `Z`). An additional character in either data type (typically `N` for DNA and `X` for AA) encodes missing data. Data can be missing for technical reasons, such as sequencing error.

When comparing phenotypic or genomic traits, it is important to ensure that only *homologous* traits are compared. Traits are homologous, if they have evolved from a common ancestor. Note that similarity alone does not necessarily imply homology. Genes or phenotypic traits can evolve independently multiple times (*convergent* evolution). For example, echolocation evolved independently in whales and bats [132].

Due to their descriptive nature, the number of traits for morphological data is orders of magnitudes smaller than the number of genotypic data characters. Therefore, modern phylogenetics typically relies on genotypic data. However, morphological data is still relevant, for example when studying the evolutionary history of fossils for which genotype data cannot be extracted.

2.1.2 Multiple Sequence Alignment

The input data for a phylogenetic inference is a set of homologous sequences representing the organisms under study. Most modern phylogenetics methods rely on a concept called *Multiple Sequence Alignment* (MSA). An MSA is a data matrix where each row corresponds to a taxon, and each column (*site*) corresponds to one trait. For morphological data, the result of data collection already is an MSA, that is, a fully aligned set of sequences, since each trait is recorded for each taxon. This is, however, not the case for genotypic data. Homologous sequences of genotypic data are generally not of the same length. Over the course of time, genomic sequences evolve as a result of substitution, insertion, or deletion events (see Figure 2.2). These changes of individual nucleotides result in genotype changes in individual organisms. Insertion and deletion (indel) events necessarily change the sequence length. Thus, before conducting any similarity analysis, we need to identify homologous characters and therefore align the sequences to each other. Computing an MSA thus requires inserting gap characters (typically denoted as `-`) such that non-gap characters that share a common evolutionary history are aligned to each other. The resulting MSA is a data matrix

$$\boldsymbol{S} = \begin{bmatrix} s_1^1 & s_1^2 & \dots & s_1^M \\ s_2^1 & s_2^2 & \dots & s_2^M \\ \dots & \dots & \dots & \dots \\ s_N^1 & s_N^2 & \dots & s_N^M \end{bmatrix} \in N \times M, \tag{2.1}$$

where N is the number of taxa, M the number of sites, and s_i^j denotes the j-th character of the i-th aligned taxon. Thus, the value of s_i^j can be either a character of the respective data type (DNA, RNA, AA), a character denoting missing data, or the gap character.

Figure 2.3 shows an exemplary MSA with four taxa and nine sites. As visualized in Figure 2.3, multiple sites can exhibit the same character composition, that is, the same site *pattern*. Hence, in addition to the number of sites, we often also characterize an MSA by the number of unique site patterns it comprises. The exemplary MSA in Figure 2.3 contains 7 patterns, since sites 3 and 5 have the same

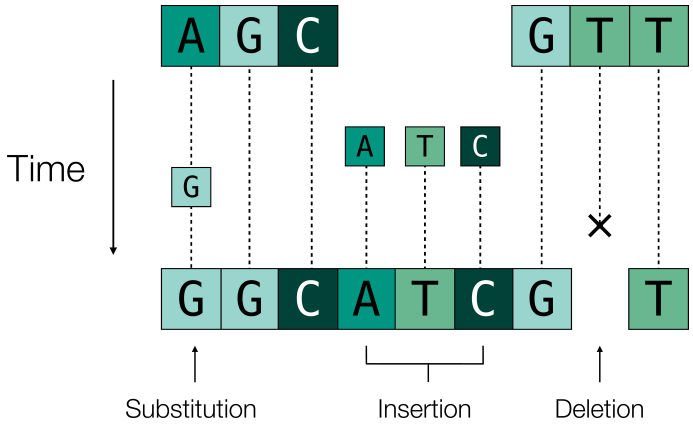


Figure 2.2: Visualization of DNA evolution events over time. Note that indel events can affect multiple subsequent nucleotides at once.

pattern `GAAC`, and sites 6 and 9 have the same pattern `CG-C`. We refer to sites that have the same character across all taxa as *invariant* sites. Site 1 in Figure 2.3 is a fully conserved, invariant site.

Site	1	2	3	4	5	6	7	8	9
Taxon 1	A	A	G	A	G	C	C	–	C
Taxon 2	A	A	A	–	A	G	C	T	G
Taxon 3	A	G	A	T	A	–	T	A	–
Taxon 4	A	A	C	–	C	C	C	–	C

Figure 2.3: Exemplary DNA MSA with four taxa and nine sites. This MSA contains one invariant site (site 1). Sites 3 and 5 (`GAAC`), and sites 6 and 9 (`CG-C`) have the same pattern, resulting in 7 unique patterns.

Computing an MSA for a given set of taxa, for example, under the sum-of-pairs (SP) score, was shown to be $\mathcal{NP}$-hard [87]. Therefore, tools for MSA inference rely on heuristic approaches [38, 88, 163]. Explaining the technical details of MSA algorithms is beyond the scope of this thesis, since we solely rely on pre-aligned data obtained from MSA databases (see Section 2.1.2.1).

Note that the properties of the MSA, such as the number of patterns and taxa, impact the subsequent phylogenetic analysis [166]. MSAs with a high *phylogenetic signal* are expected to be easier to analyze under the $\mathcal{NP}$-hard ML criterion, since the space of potential trees can be explored more systematically due to the higher information content. Consequently, multiple, independently inferred trees on such an MSA are expected to be highly similar in terms of their topology. In general, the phylogenetic signal for MSAs with few taxa and either many or longer genes, or entire genomes, is stronger than in MSAs with many taxa and fewer and/or shorter genes [166].

2.1.2.1 MSA Databases

Different MSA heuristics potentially result in different MSAs for the same set of sequences [39]. Thus, researchers typically publish the MSA alongside their studies. A popular database for sharing MSAs is TreeBASE [137]. In this thesis, we use a snapshot of approximately 10 000 MSAs that we obtained from TreeBASE in January 2022, including DNA, AA, and morphological data. This data collection comprises MSAs that have been analyzed in the context of published phylogenetic studies. It thus provides a representative collection of typical phylogenetic analyses.

We also use DNA and AA MSAs obtained from the RAxML Grove database [79]. The publicly available RAxML Grove database contains phylogenetic trees based on fully anonymized MSAs that were analyzed using the RAxML-NG [96] and RAxML [167] web-servers. Note that we do not make the MSAs publicly available, as they have not necessarily been published (or were not intended for publication) by the user of the web-servers (yet). These fully anonymized MSAs are thus only available internally within our research group. Note that for our work, we rely solely on summary statistics, such as the number of sites or taxa, of these anonymous MSAs. These summary statistics preserve the anonymity of the underlying data, as they do not allow for any reverse-engineering of the biological content of the respective MSAs.

Both, the TreeBASE data, and the RAxML Grove data contain diverse MSAs without a specific focus on the type of underlying taxa or traits.

The majority of MSAs in our TreeBASE data collection, as well as in RAxML Grove, are DNA MSAs. To increase the set of AA MSAs, we also use on the HOGENOM database [136] in this thesis. The HOGENOM database consists of sequences obtained from whole-genome data of Bacteria, Archaea, and Eukarya, and thus also comprises a diverse sample of empirical genomic data.

2.1.3 Phylogenetic Tree Inference

Over the course of the last decades, researchers have developed a plethora of tree inference methods that mostly rely on MSAs. While alignment-free methods are generally simpler, they have been shown to be less accurate than MSA-based approaches [78]. MSA-based methods can be divided into two categories: distance-based methods and character-based methods. Distance-based methods directly rely on the assumption that higher sequences similarities indicate a closer evolutionary relationship between the respective taxa. Implementations of distance methods, such as Neighbor-Joining [150] or UPGMA [164] infer a phylogenetic tree using pairwise distances between the sequences in the MSA. In contrast, character-based methods score trees for a given MSA under an explicit optimality criterion. For example, the Maximum Parsimony (MP) method [44, 52] scores trees by counting mutations, and the Maximum Likelihood (ML) method [47] deploys a statistical model of sequence evolution to compute the likelihood of observing a tree given an MSA. Finding the globally optimal tree under such a criterion requires scoring of all possible unrooted binary tree topologies. Such an exhaustive tree search is computationally infeasible

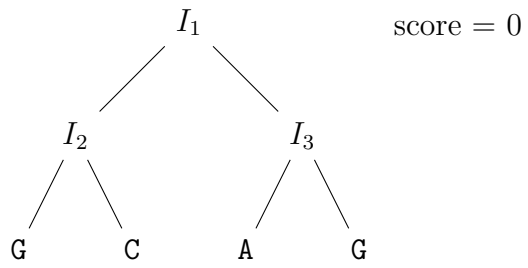
due to the large number of possible tree topologies. Given a set of $N \geq 3$ taxa, the number of possible unrooted trees is $\prod_{i=3}^{N} 2i - 5$ [49].

Many popular character-based tree inference methods, including ML and MP, were shown to be $\mathcal{NP}$-hard [31, 55]. Therefore, their implementation relies on heuristic search strategies. Such heuristics typically initiate the tree search on a set of one or multiple candidate trees obtained, for example, via distance-based methods, which are then further optimized under the character-based criterion, until a convergence criterion is reached. However, these heuristics are not guaranteed to find the globally optimal solution and different optimization strategies can lead to different (locally optimal) solutions. In this thesis, we use ML and MP methods for tree inference and rely on the respective heuristics implemented in RAxML-NG [96]. In the following sections, we explain both criteria (ML and MP), and the respective heuristic implementations in RAxML-NG in greater detail.

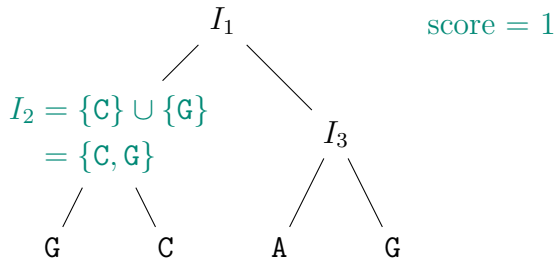
2.1.3.1 Maximum Parsimony

With the Maximum Parsimony (MP) method [44, 52], the goal is to find the phylogenetic tree that explains the MSA $\boldsymbol{S}$ with the least number of mutations. For a given topology, the parsimony score can be computed in $\mathcal{O}(N \cdot M)$ operations using dynamic programming. The MP method computes the parsimony score for each site separately, relying on the assumption that sites evolve independent of each other. The overall parsimony score for a given topology and the respective MSA $\boldsymbol{S}$ is the sum over all per-site scores. Using Fitch's algorithm [52], we can compute the per-site score as follows. We initially place a virtual root in the tree. Note that this is only a technical requirement, and the placement of the root does not change the parsimony score. We then traverse the tree bottom-up from the tips to the root. The tip states are the observable data and correspond to the states of the taxa represented by their sequences in the underlying MSA. The inner states of the tree are unknown. In each step of the parsimony score computation, we assign a set of potential characters to these inner states. To determine the state of an inner node I_i, we first compute the intersection of its two descendant character sets l and r. An empty intersection indicates a mutation event, as there is no consensus between the two descendants. In this case, we increment the parsimony score by 1 and assign the union of l and r to the inner state I_i ($I_i = l \cup r$). If the intersection is non-empty, there exists a possible ancestral state that can explain this subtree rooted at I_i without any mutation. Hence, we do not need to increment the parsimony score. In this case, we set the inner state I_i to the intersection of l and r ($I_i = l \cap r$).

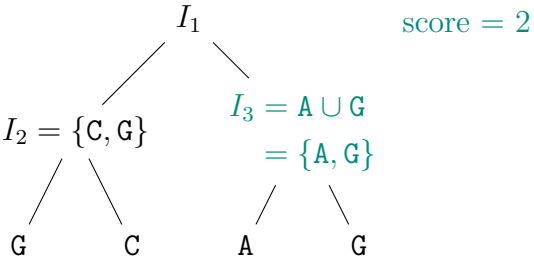
The following example visualizes the parsimony score computation using an exemplary tree topology for a single site in a DNA MSA. We place the virtual root I_1 into the branch connecting I_2 with I_3 and traverse the tree bottom up. The initial parsimony score is 0.



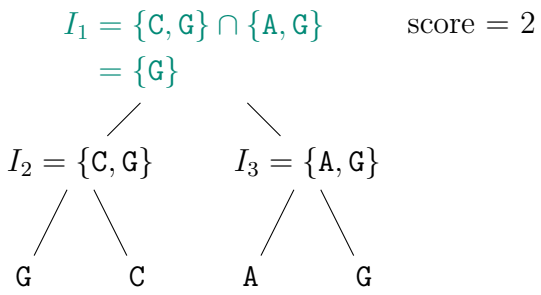
The first step is to compute the set of inner states for the inner node I_2. The intersection of its child nodes G and C is empty. Thus, we assign the state of I_2 as the union of the descendant states $I_2 = \{\mathtt{C}\} \cup \{\mathtt{G}\} = \{\mathtt{C}, \mathtt{G}\}$ and increment the parsimony score by 1.



Analogously, we set $I_3 = \mathtt{A} \cup \mathtt{G} = \{\mathtt{A}, \mathtt{G}\}$ and increment the parsimony score by 1 to account for this mutation event, thus obtaining a total score of 2.



The final step is to determine the inner state of the virtual root I_1. We first compute the intersection of its descendants I_1 and I_2 as $I_1 \cap I_2 = \{\mathtt{C}, \mathtt{G}\} \cap \{\mathtt{A}, \mathtt{G}\} = \{\mathtt{G}\}$. We notice that the intersection is non-empty, and set $I_3 = \{\mathtt{G}\}$. The non-empty intersection indicates that there exists a possible assignment for I_1 that explains the data in I_2 and I_3 without any mutation (the nucleotide G in this case). Hence, we do not need to increase the parsimony score.



Since we have reached the virtual root, the computation of the parsimony score terminates, and we determined the overall parsimony score for this example tree topology as 2. Using the inner states we computed in the tree, we can also reconstruct the ancestral states using a top-down approach. As this is not relevant for merely computing the parsimony score of a given tree, we do not further explain this procedure and refer the interested reader to the original description of Fitch's algorithm [52].

While scoring a tree under the MP criterion is simple and fast, *generating* the most-parsimonious tree (that is, the tree with the least mutations) is not. As stated above, enumerating each possible topology and computing its score is computationally infeasible due to the vast number of potential tree topologies. The MP problem was shown to be $\mathcal{NP}$-hard [55] and exploring the space of possible tree topologies under the MP criterion requires heuristic algorithms that rely on tree topology optimization strategies (see Section 2.1.3.3). Such heuristics are comparatively fast due to the relatively simple model of parsimonious evolution, they are, however, less accurate than other methods with more complex (statistical) models of evolution [130]. For example, MP cannot account for multiple unobserved substations along a single branch (for example, $\mathtt{T} \rightarrow \mathtt{C} \rightarrow \mathtt{G}$). Additionally, MP is prone to *long-branch attraction* (LBA) [15]. LBA is an estimation bias induced by rapidly evolving lineages, resulting in long branches. These lineages are inferred as being closely related because of their similarly high evolutionary rates.

2.1.3.2 Maximum Likelihood

The Maximum Likelihood (ML) method relies on a statistical model of evolution and strives to find the most likely tree among all possible trees under such a model. Thus, ML is an optimization problem that optimizes the likelihood $L(\boldsymbol{\theta}|\boldsymbol{S})$ for a set of parameters $\boldsymbol{\theta}$ given the data $\boldsymbol{S}$. This corresponds to the probability of observing $\boldsymbol{S}$ given the parameters $\boldsymbol{\theta}$

$$L(\boldsymbol{\theta}|\boldsymbol{S}) = P(\boldsymbol{S}|\boldsymbol{\theta}). \tag{2.2}$$

In ML-based phylogenetic inference, the parameter set $\boldsymbol{\theta} = (T, \boldsymbol{b}, \boldsymbol{M}, \phi)$ comprises the tree topology T, the corresponding branch lengths $\boldsymbol{b}$, and the substitution model $\boldsymbol{M}$ with additional parameters ϕ.

To simplify the following explanation of ML based phylogenetic inference, we will focus on DNA data with four states corresponding to the four nucleotides $\mathtt{A}$, $\mathtt{C}$, $\mathtt{G}$, and $\mathtt{T}$. The principle translates directly to all other data types presented in Section 2.1.1 using the respective character set.

Phylogenetic Likelihood Computation

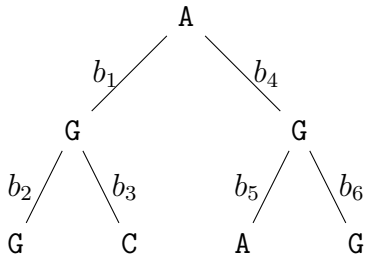
The ML method is not a generative method. That means it does not propose parameters, but rather evaluates a given set of parameters $\boldsymbol{\theta}$ via the phylogenetic likelihood. Thus, in addition to the MSA, we require a candidate tree topology (a *starting tree*) including branch lengths, and a set of model parameters to compute the respective likelihood. As shown in Equation (2.2), we compute the phylogenetic likelihood $L(\boldsymbol{\theta}|\boldsymbol{S})$ as the probability $P(\boldsymbol{S}|\boldsymbol{\theta})$ of observing $\boldsymbol{S}$ given the current values of parameters in $\boldsymbol{\theta}$. To simplify this computation, in phylogenetics, we assume that sites evolve independently. We can hence reformulate the phylogenetic likelihood as

$$L(\boldsymbol{\theta}|\boldsymbol{S}) = \prod_{i=1}^{M} P(s_i|\boldsymbol{\theta}), \tag{2.3}$$

where s_i denotes the i-th column in the MSA. The per-site probabilities $P(s_i|\boldsymbol{\theta})$ can become very small, and potentially induce numerical issues. To prevent underflow, we compute the logarithm of the likelihood instead:

$$log\left(L(\boldsymbol{\theta}|\boldsymbol{S})\right) = \sum_{i=1}^{M} log\left(P(s_i|\boldsymbol{\theta})\right). \tag{2.4}$$

To simplify the following explanation of the per-site likelihood calculation $L(\boldsymbol{\theta}|s_i)$ for site i, we consider the following exemplary tree topology for a single site in a DNA MSA:



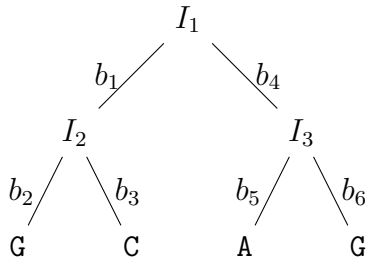
We compute the likelihood for this site as the product of the transition probabilities from parent to child nodes and the probability of observing nucleotide `A` at the root of the tree.

$$\begin{aligned} L(\boldsymbol{\theta}|s_i) = P(s_i|\boldsymbol{\theta}) = \pi_{\mathtt{A}} \cdot P_{\mathtt{A}\to\mathtt{G}}(b_1) \cdot P_{\mathtt{G}\to\mathtt{G}}(b_2) \cdot P_{\mathtt{G}\to\mathtt{C}}(b_3) \\ \cdot P_{\mathtt{A}\to\mathtt{G}}(b_4) \cdot P_{\mathtt{G}\to\mathtt{A}}(b_5) \cdot P_{\mathtt{G}\to\mathtt{G}}(b_6). \end{aligned} \tag{2.5}$$

The transition probabilities $P_{i\to j}$ and the prior probability $\pi_{\mathtt{A}}$ are given by the probabilistic model of sequence evolution $\boldsymbol{M}$. We will provide more details on the model of

evolution below. The branch lengths b_i are proportional to the relative evolutionary time between the respective nodes, with $t_i = r \cdot b_i$ where r denotes the evolutionary rate (see below).

In contrast to this example, in phylogenetics, we typically only know the states of the terminal nodes (the taxa) as the inner states are unknown:



Consequently, when computing the per-site likelihood, we need to account for all possible combinations of inner, ancestral states and the per-site likelihood computation changes to

$$L(\boldsymbol{\theta}|s_i) = \sum_{I_1} \sum_{I_2} \sum_{I_3} \pi_{I_1} \cdot P_{I_1 \to I_2}(b_1) \cdot P_{I_2 \to \mathtt{G}}(b_2) \cdot P_{I_2 \to \mathtt{C}}(b_3) \cdot P_{I_1 \to I_3}(b_4) \cdot P_{I_3 \to \mathtt{A}}(b_5) \cdot P_{I_3 \to \mathtt{G}}(b_6), \tag{2.6}$$

with $I_1, I_2, I_3 \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. So essentially, we sum over the probabilities for all possible combinations of ancestral states I_1, I_2, and I_3.

This computation can be efficiently implemented using the Felsenstein Pruning algorithm [46]. Similar to the MP problem, ML is $\mathcal{NP}$-hard [31] and phylogenetic inference tools implementing the ML method rely on heuristic parameter optimization strategies (see Section 2.1.3.3).

Model of Evolution

The probabilistic model of sequence evolution plays a crucial role in the likelihood computation. We model sequence evolution as a continuous-time Markov chain, where the current state depends only on the previous state. Each possible character of the respective data type corresponds to one state in the Markov chain, and state transitions correspond to character substitutions. This substitution process is modelled as a matrix $\boldsymbol{Q}$, where $Q_{i \to j}$ denotes the instantaneous *rate* of substitution from state i to state j. The rate $Q_{i \to i}$ of remaining in state i is obtained by the constraint that all rows in $\boldsymbol{Q}$ must sum to 0. To obtain the transition *probability*, we need to generalize $\boldsymbol{Q}$ for any time $t > 0$. This is modelled via the transition probability matrix

$$\boldsymbol{P}(t) = e^{\boldsymbol{Q}t}, \tag{2.7}$$

where $P_{i \to j}(t)$ is the probability of transitioning from state i to state j in time t. Based on $\boldsymbol{P}$, we can model multiple substitutions along a single branch over time $t_1 + t_2$ using the Chapman–Kolmogorov equation as a sum over all possible intermediate states between i and j

$$P_{i \to j}(t_1 + t_2) = \sum_k P_{i \to k}(t_1) P_{k \to j}(t_2). \tag{2.8}$$

When inferring unrooted phylogenetic trees, we assume time-reversibility. This means that the probability of starting in state i and evolving into state j over time t is identical to the probability of starting in state j and evolving into state i over time t. Consequently, the choice of the root of a tree is arbitrary, as we do not assume a direction of ancestral relationships. Non-reversible models can be used to determine the root of an unrooted tree, as the likelihood under such a model is affected by the root placement [18].

For time-reversible models, we obtain the (symmetric) rate matrix $\boldsymbol{Q}$ as

$$\boldsymbol{Q} = \boldsymbol{R} \cdot diag(\boldsymbol{\Pi}), \tag{2.9}$$

where $\boldsymbol{\Pi}$ denotes the *equilibrium frequencies* and $\boldsymbol{R}$ the substitution rates between states. The equilibrium frequencies are the prior probabilities of observing each state. For DNA data, $\boldsymbol{\Pi} = (\pi_{\mathtt{A}}, \pi_{\mathtt{C}}, \pi_{\mathtt{G}}, \pi_{\mathtt{T}})$ where $\pi_{\mathtt{A}}$ denotes the prior probability of observing the nucleotide `A`.

The most general time-reversible DNA evolution model is the *General Time Reversible* (GTR) model [173] with 10 parameters

$$\boldsymbol{R} = \begin{pmatrix} \alpha & & \\ \beta & \gamma & \\ \delta & \epsilon & \zeta \end{pmatrix} \quad \boldsymbol{\Pi} = (\pi_{\mathtt{A}}, \pi_{\mathtt{C}}, \pi_{\mathtt{G}}, \pi_{\mathtt{T}}). \tag{2.10}$$

Given the constraints that $\sum_i \pi_i = 1$, and using relative rates, the GTR model has eight free parameters.

Other, simpler models can be derived from the GTR model by restricting the parameters. For example, the *HKY* model [70] allows for distinct equilibrium frequencies, but only emulates two types of nucleotide substitutions: transitions ($\mathtt{A} \leftrightarrow \mathtt{G}$, $\mathtt{T} \leftrightarrow \mathtt{C}$) and transversions ($\{\mathtt{A}, \mathtt{G}\} \leftrightarrow \{\mathtt{T}, \mathtt{C}\}$). The simplest model of DNA evolution, the *Jukes-Cantor* (JC) model [85], restricts the rate matrix to a single rate parameter for all substitutions and defines the equilibrium frequencies as $\pi_{\mathtt{A}} = \pi_{\mathtt{C}} = \pi_{\mathtt{G}} = \pi_{\mathtt{T}} = 1/4$.

The most basic model of AA evolution is the *Poisson* model, with equal AA substitution rates and equal equilibrium frequencies ($\pi_i = 1/20$). In analogy to the GTR model, the *GTR20* model is the most flexible model of AA evolution, with 208 free parameters (189 for the relative AA substitution rates and 19 free parameters for

estimating the 20 equilibrium frequencies). Such a parameter-rich model can lead to unstable parameter estimates, especially if the MSA $\boldsymbol{S}$ has few sites and taxa. Thus, for AA data, it is common to use pre-computed substitution matrices that have been derived from large empirical data collections, such as the *WAG* [186] or the *LG* model [103]. The LG model is expected to better represent evolution than the WAG model, as it was derived from a larger and more diverse set of AA MSAs and is based on a more refined rate inference technique.

Over the past decades, a plethora of distinct evolutionary models, both for DNA and for AA data, have been proposed. For a comprehensive overview of commonly used models in phylogenetics, we refer the interested reader to the documentation of the popular phylogenetic inference toolkit IQ-TREE[1] [121].

The models discussed so far assume a constant rate r of evolution along the entire MSA. This, however, is not true for most empirical data. Different MSA sites experience distinct evolutionary pressure, for instance, due to their structure or functionality [193]. We refer to this phenomenon as *among site rate heterogeneity*. The most fundamental method to account for rate heterogeneity is to assume that a certain proportion of sites is constant or invariant over evolutionary time, meaning that all taxa have the same character at this particular site. The proportion of invariant sites is typically estimated based on the MSA $\boldsymbol{S}$. Throughout this thesis, when referring to a substitution model that estimates this proportion of invariant sites, we add the suffix '+I' to the model's name. For example, we refer to the GTR model with an additional free parameter for the proportion of invariant sites as GTR+I.

This simple model of rate heterogeneity only distinguishes between conserved and evolving sites. A more elaborate method models the rate of evolution as a statistical distribution, typically via a Γ distribution [192]. The Γ distribution has two parameters, a *shape* parameter $\alpha > 0$ and a *scale* parameter $\beta > 0$. To reduce the complexity, both parameters are set to the same value $\beta := \alpha$ [194]. The degree of rate heterogeneity is inversely related to the shape parameter α. For high rate heterogeneity ($\alpha \leq 1$), the Γ distribution is left-skewed, meaning that most sites evolve relatively slow with a few substitution hotspots having high evolutionary rates. For low rate heterogeneity ($\alpha > 10$), the Γ distribution is bell-shaped around a mean of 1, and only a few sites evolve at a faster or slower pace. The scale parameter α is estimated based on the MSA $\boldsymbol{S}$. Figure 2.4 shows the probability density functions of Γ distributions for three different α values.

In practice, the continuous Γ distribution is approximated using a discrete distribution with K rate categories. The per-site likelihood is then computed as the average of the per-rate likelihoods over the respective site

$$L_\Gamma(\boldsymbol{\theta}|s_i) = \frac{1}{K}\sum_{u=1}^{K} L(\boldsymbol{\theta}|s_i, r_u), \tag{2.11}$$

[1] http://www.iqtree.org/doc/Substitution-Models

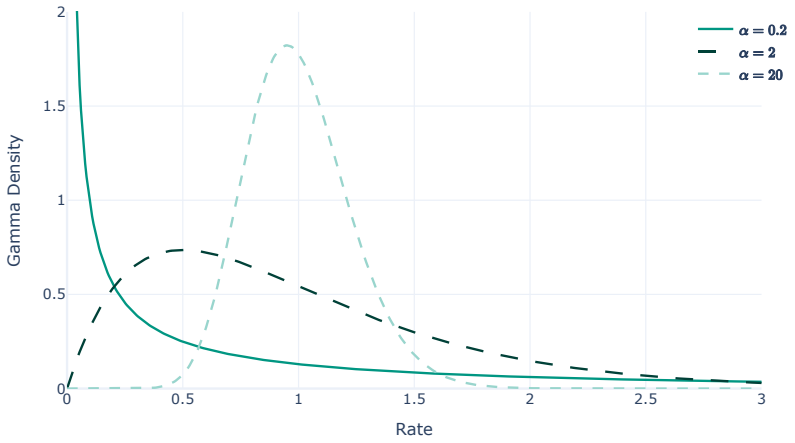


Figure 2.4: Probability density function of the Γ distribution for different scale parameters α with $\beta := \alpha$. The x-axis shows the evolutionary rate, and the y-axis is proportional to the number of sites with that evolutionary rate.

where r_u denotes the evolutionary rate for the u-th category.

The computational time and space complexity of the likelihood computation increases linearly with the number of discrete categories used to approximate the continuous Γ distribution. As a tradeoff between accuracy and computational overhead, following the suggestion by Yang [192], typically a default of $K := 4$ categories is used. If not stated otherwise, throughout this thesis, we use this default setting of four discrete rates for all models using a Γ distribution to model rate heterogeneity. We denote the respective model with a suffix '+G' (for example, GTR+G).

Evolutionary pressure and hence heterogeneity does not only affect the per-site rate of evolution within a gene, but also impacts the individual substitution rates $Q_{i \to j}$ [105]. General substitution matrices, such as the GTR or the WAG models, fail to capture such patterns. Inferring a distinct model per site in the MSA is statistically infeasible due to the large number of parameters. Instead, *mixture models* estimate site-dependent substitution rates using a set of K distinct substitution models $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_K$ with corresponding distinct rate matrices $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_K$. These individual models can reflect different site categories, for example, coding sites (sites encoding a gene) or buried sites (sites located in the interior of the three-dimensional protein structure) [105]. The assignment of individual sites to these categories is typically unknown or only known with some uncertainty. Therefore, the per-site likelihood under a mixture model is computed as the weighted average over the per-site likelihood under each separate model:

$$L_M(\boldsymbol{\theta}|s_i) = L_M((T, \boldsymbol{b}, \boldsymbol{M_M}, \phi_M)|s_i) = \frac{1}{K} \sum_{u=1}^{K} w_u L_M((T, \boldsymbol{b}, \boldsymbol{M_u}, \phi_u)|s_i). \tag{2.12}$$

The weight w_u is the *a priori* probability of the respective substitution model $\boldsymbol{M_u}$ and quantifies the probability of site s_i having evolved under the substitution model

$\boldsymbol{M_u}$. Mixture models are typically based on empirical data collections. For example, for the AA *C60* mixture model, Le *et al.* [104] determined 60 distinct equilibrium frequency profiles and their respective weights from over 1000 high-quality empirical AA MSAs using an expectation–maximization algorithm. The more recent *UDM* models [153] provide up to 256 distinct equilibrium frequency profiles based on over 2000 AA MSAs obtained from the HOGENOM [136] and HSSP [152] databases.

When collecting data for phylogenetic analyses, we often concatenate multiple genes into a single MSA. Similar to differences in per-site evolutionary rates, genes evolve at different rates as they underlay different evolutionary pressures. To account for these differences at a larger scale, we can organize such MSAs into multiple *partitions* (typically by genes) and estimate a distinct set of model parameters for each partition independently. We refer to such MSAs as *partitioned MSAs*.

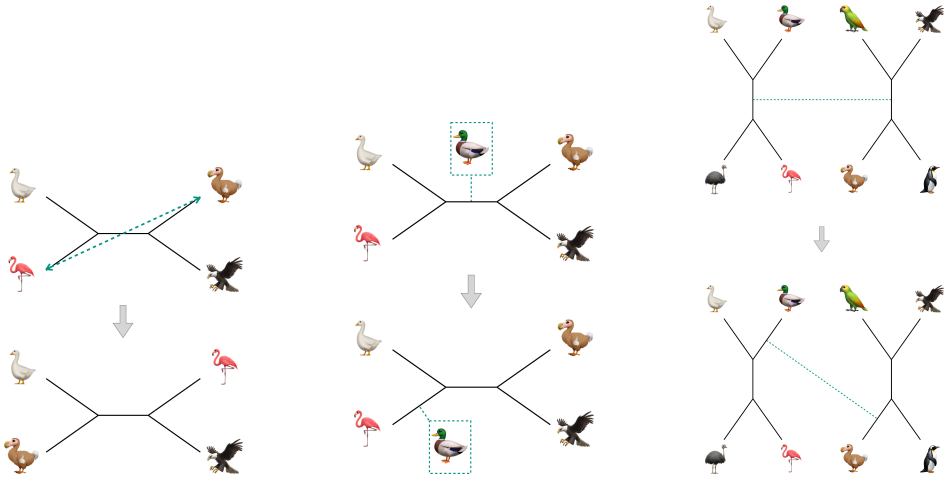
2.1.3.3 Parameter Optimization

As stated above, both the MP method, and the ML method are not generative methods and only evaluate candidate parameters. Implementing these criteria for tree *inference* thus relies on optimization routines to improve upon a given candidate tree topology and, under ML, branch lengths and substitution model parameters. In the following paragraphs, we outline different parameter optimization strategies for topology, branch length, and model parameter optimization.

Topology Optimization

The tree topology can be optimized by rearranging a given candidate topology. The most common strategies are Nearest Neighbor Interchange (NNI), Subtree Pruning and Regrafting (SPR), and Tree Bisection and Reconnection (TBR) (Figure 2.5) [194]. The main difference between these three methods is their complexity in terms of the number of potential rearrangement moves and, consequently, their ability to explore the space of possible tree topologies. The most complex approach is TBR, with SPR being a subset of TBR, and NNI being a subset of SPR.

Each inner branch of a binary, unrooted tree connects four subtrees. Note that a subtree can also be a terminal node. With the NNI strategy, these four subtrees are detached and rearranged, resulting in two possible alternative topologies. For a tree with N taxa, the number of possible NNI moves is in $\mathcal{O}(N)$. Figure 2.5(a) visualizes one possible NNI move for a tree with four taxa. The SPR strategy detaches a single subtree from the tree (pruning) and re-attaches (regrafts) it to another branch in the tree. The number of possible SPR moves for a tree with N taxa is in $\mathcal{O}(N^2)$, as there are $\mathcal{O}(N)$ possible subtrees to prune and $\mathcal{O}(N)$ possible regrafting positions in the tree. Figure 2.5(b) visualizes one possible SPR move for a five-taxon tree. The computationally most costly strategy is TBR. TBR generates candidate topologies by removing a single inner branch from the tree and reconnecting each possible pair of branches in the two subtrees. The number of possible moves is in $\mathcal{O}(N^3)$ as there are $\mathcal{O}(N)$ possible inner branches to remove, and $\mathcal{O}(N)$ possible branches for reconnection in each of the two subtrees. Figure 2.5(c) visualizes one possible TBR move for a tree with eight taxa.



(a) NNI **(b)** SPR **(c)** TBR

Figure 2.5: Schematic explanation of topology optimization strategies.

Choosing the best topology optimization strategy constitutes a trade-off between efficiency in terms of runtime, and efficiency in terms of sampling the tree space of potential alternative topologies. While NNI moves are computationally cheap, getting stuck in a local optimum is more likely than with the computationally more costly TBR strategy due to the lower number of neighboring topologies that can be generated, and thus, a less exhaustive sampling of the tree space.

Branch Length and Model Parameter Optimization

Branch lengths and substitution model parameters can be iteratively optimized using standard gradient-based numerical routines. The goal is to find a parameter ζ that maximizes the likelihood $L(\zeta)$. To find extreme values of this function, we need to find roots of the derivative, meaning we need to find ζ such that $L'(\zeta) = 0$. The Newton-Raphson method is frequently used for branch length optimization, Brent's algorithm [24] and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [53] are commonly used to determine optimal substitution model parameters. The equilibrium frequencies can be obtained by either counting the respective frequencies in the MSA $\boldsymbol{S}$, or a similar numerical optimization approach as for the substitution rates.

2.1.3.4 RAxML-NG

Different tree inference tools implement distinct combinations of optimization strategies to find the *best-known* ML or MP tree. That is, the tree with the highest score according to the respective criterion. Over the course of the past decades, numerous tools have been developed. Popular examples for ML tree inference include RAxML [167], its successor RAxML-NG [96], IQ-TREE [121], or FastTree [142]. Further improving upon existing heuristics, both in terms of finding a better best-known tree, and improving upon the runtimes, is the subject of ongoing research. In this thesis, we use RAxML-NG for both, MP, and ML tree inference. In this section, we briefly explain the respective RAxML-NG heuristics. The explanations

are based on Kozlov *et al.* [96], Stamatakis *et al.* [169], and Stamatakis and Kozlov [168].

Maximum Parsimony

To infer a tree under the MP criterion, RAxML-NG generates a tree using *randomized stepwise sequence addition*. First, RAxML-NG selects three taxa t_1, t_2, t_3 at random from the set of $N \geq 3$ taxa and builds the corresponding single possible unrooted, binary tree topology. It then chooses another taxon t_i from the remaining set of $N-3$ taxa uniformly at random and inserts it into every possible branch in the existing topology. Each placement is scored under the MP criterion, and finally t_i is inserted into the branch with the best parsimony score. This procedure is repeated until all taxa have been placed into the tree.

Maximum Likelihood

RAxML-NG implements a greedy hill-climbing procedure to find the best-known ML tree. This means that only optimization steps improving the likelihood are being accepted. The ML tree inference is initialized using candidate tree topologies (starting trees) containing all taxa of the input MSA. These starting trees can either be random trees, meaning that the taxa are randomly combined into an unrooted, binary topology, or they can be generated using the MP-based randomized stepwise sequence addition as outlined above. By default, a single RAxML-NG tree inference run uses 10 random and 10 MP starting trees and yields 20 ML trees. However, both, the total number of generated trees, and the number of random and MP starting trees can be adjusted by the user. RAxML-NG optimizes the tree topology using SPR moves. To alleviate the computational complexity of SPR moves, RAxML-NG only regrafts branches up to a given maximum radius around the pruning position. This radius is automatically determined during the first round of SPR topology optimization moves, or can be defined by the user. Branch lengths are optimized using the Newton-Raphson method, and the BFGS and Brent's method are deployed for optimizing the substitution model parameters, the equilibrium frequencies, and the α shape parameter of the Γ distribution.

2.1.4 Comparing Phylogenetic Trees

Due to the heuristic nature of MP and ML tree inference approaches, we usually infer multiple phylogenies for the same MSA using distinct starting points for each inference. Before drawing conclusions about the underlying evolutionary process of the data, we need to compare the inferred trees and identify conclusive results. We typically compare trees using topological distance metrics and filter results for conclusive trees using statistical significance tests that compare the likelihoods of the inferred trees. With both, distance metrics and statistical tests, it is important to note that we can only compare trees comprising the same set of taxa.

2.1.4.1 Distance Metrics

A plethora of metrics exists to compute the topological distance between phylogenetic trees. One of the most commonly used metrics is the *Robinson-Foulds Distance* (RF-Distance) [145]. The RF-Distance is simple and fast to compute, and thus useful for the pairwise comparison of large tree sets. However, it can be insensitive to small topological changes. In such cases, the more fine-grained *Quartet distance* can be used [42]. The Quartet distance counts the number of differing four-taxa subtrees (*quartets*). However, compared to the RF-Distance, normalizing the Quartet distance is not straightforward [36]. When inferring trees using the ML criterion, we obtain trees with optimized branch lengths. Both the RF-Distance and the Quartet distance only compare the topologies of the inferred trees. Consequently, two trees with the same topology but differing branch lengths will have a distance of 0. In contrast, the *branch score* (also referred to as the *Kuhner-Felsenstein distance*) [98] additionally accounts for differences in branch lengths when comparing phylogenies. In this thesis, we rely on the RF-Distance, as we are mainly interested in topological differences between large sets of inferred trees for our applications.

In the following, we briefly explain the computation of the RF-Distance between two trees T_1 and T_2. First, we need to define *internal* and *external* branches, as well as *bipartitions*. An external branch connects a leaf to the tree, whereas an internal branch connects two inner nodes. Each branch in the tree splits the set of taxa into two disjoint subsets and thus induces one *bipartition*. External branches induce *trivial* bipartitions. These trivial bipartitions are non-informative because any tree with the same set of taxa induces these bipartitions. Using the sets of non-trivial bipartitions B_1 and B_2 induced by T_1 and T_2 respectively, the RF-Distance is defined as

$$d(T_1, T_2) = |B_1 \cup B_2| - |B_1 \cap B_2|. \tag{2.13}$$

We can normalize the RF-Distance using the maximum possible distance between T_1 and T_2. A binary tree with N taxa has $N - 3$ internal branches. If all non-trivial bipartitions induced T_1 and T_2 differ, the RF-Distance is $2(N-3)$. Thus, the normalized RF-Distance is defined as

$$RF(T_1, T_2) = \frac{d(T_1, T_2)}{2(N-3)}. \tag{2.14}$$

Note that throughout this thesis, we use the *normalized* RF-Distance and denote it as RF-Distance for the sake of simplicity.

2.1.4.2 Statistical Tests

As stated above, when inferring phylogenies under the ML criterion, we initiate multiple distinct tree inference procedures that result in – potentially highly – distinct *locally* optimal *candidate* trees $\mathcal{T}$. Due to the heuristic nature of ML tree inference procedures, we cannot simply select the tree with the highest likelihood among these

candidate trees and define it as the best hypothesis. Instead, we have to determine if the likelihoods of the inferred trees are significantly different from each other given the underlying distribution of likelihoods. However, since we cannot re-run the process of evolution, we do not know the "true" distribution of likelihoods and therefore need to estimate it to perform statistical significance tests. A common method to estimate distributions given a limited set of observations is the *bootstrap*. In phylogenetics, the most common procedure for computing statistical confidence is the non-parametric *Felsenstein bootstrap* [48]. A single bootstrap replicate $\boldsymbol{S}^{(i)}$ is obtained by resampling the sites of the original MSA $\boldsymbol{S}$ with replacement to obtain a new MSA of the same length, but with a distinct site composition. This process is repeated to obtain a set of bootstrap replicates $\mathcal{B}$. For likelihood-based statistical testing, for each candidate tree $T_x \in \mathcal{T}$ the branch lengths and substitution model parameters need to be re-optimized based on each obtained bootstrapped MSA $\boldsymbol{S}^{(i)} \in \mathcal{B}$ to obtain the respective maximum likelihood value $L_x^{(i)}$. Additionally, the likelihoods need to reflect the expected distribution under the null hypothesis of the respective statistical test. The standard approach to adjust the likelihood values to conform to this requirement is *centering*: the likelihood value $L_x^{(i)}$ of candidate tree T_x under bootstrap replicate $\boldsymbol{S}^{(i)}$ is shifted by the mean likelihood $\bar{L}_x = \frac{1}{B}\sum_{i=1}^{B} L_x^{(i)}$ to obtain the centered likelihood $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x$. This repeated re-optimization of branch lengths and substitution model parameters is very time- and resource-consuming. Instead, Kishino *et al.* [92] suggested the *resampling estimated log-likelihoods* (RELL) method as an approximation of the non-parametric Felsenstein bootstrap. As discussed in Section 2.1.3.2, the phylogenetic likelihood is computed as the sum over the per-site log-likelihoods. The RELL bootstrap procedure resamples these per-site log-likelihoods to quickly approximate likelihoods of trees optimized under bootstrapped MSAs, thus allowing to approximate the underlying distribution as a basis for likelihood-based statistical tests.

In this thesis, we rely on six likelihood-based statistical tests implemented in the phylogenetic inference toolkit IQ-TREE [121], which we will briefly explain in the following. IQ-TREE implements the RELL approximation of the standard bootstrap as a basis for the statistical tests. Throughout this thesis, when applying statistical tests, we rely on 10 000 RELL samples.

Kishino-Hasegawa Test

The Kishino-Hasegawa (KH) [91] test compares two candidate trees T_1 and T_2 by comparing their likelihood difference $\delta = L_1 - L_2$ to the expected distribution of likelihood differences approximated using the RELL method outlined above. If both trees are equally well-supported, we expect $L_1 = L_2$ and thus

$$H_0 : E[\delta] = 0$$
$$H_A : E[\delta] \neq 0.$$

The KH test rejects the null hypothesis if the test statistic δ falls outside the acceptance region of $1 - \alpha$. The KH test relies on candidate trees to be chosen *a priori*,

that is, independently of any analysis of the underlying data. However, in phylogenetics, candidate trees are chosen *a posteriori*, for example, as a result of ML tree inferences. Goldman *et al.* [60] thus conclude that the null hypothesis $E[\delta] = 0$ is not reasonable and a one-sided KH test is more appropriate when comparing multiple candidate trees against the tree T_{ML} with the highest log-likelihood L_{ML} in the candidate tree set.

Shimodaira-Hasegawa Test

The Shimodaira-Hasegawa (SH) [159] test is a statistical test specifically designed for comparing multiple *a posteriori* selected candidate trees. The respective test hypotheses are

$$H_0 : \text{all } T_x \in \mathcal{T} \text{ are equally well-supported}$$
$$H_A : \text{some or all } T_x \in \mathcal{T} \text{ are not equally well-supported.}$$

Instead of pairwise comparisons, each candidate tree T_x is compared against T_{ML}, the tree with the highest maximum likelihood value in $\mathcal{T}$. Thus, the test statistic for candidate tree T_x is $\delta_x = L_{ML} - L_x$. Similar to the KH test, the respective distribution under the null hypothesis is approximated using the RELL bootstrap procedure and subsequent centering of the likelihoods. Since each candidate tree is compared against the ML tree, the SH test is a one-sided test and, for a 5% significance level, H_0 is rejected if the test statistic δ_x is in the rejection area beyond 95%. The SH test only yields the correct estimations of significance levels if the *globally* optimal ML tree is in the candidate tree set [151]. Strimmer and Rambaut [171] additionally showed that the SH test is biased by the size of the candidate tree set and is more conservative when the number of trees in the candidate tree set increases.

The weighted variants of the KH and SH test scale the likelihood differences using the variance of differences in the set of candidate trees $\mathcal{T}$. The idea of this weighting is to obtain a less conservative test statistic [151].

Approximately Unbiased Test

The SH test overestimates the selection bias and provides more conservative significance levels with an increased number of candidate trees. To correct for this selection bias, Shimodaira [158] proposed the Approximately Unbiased (AU) test. The AU test is based on a *multiscale bootstrap* procedure. The standard bootstrap procedure generates replicates with the same number of sites M as the original MSA. Instead, the multiscale bootstrap, generates replicates with different lengths $M_r = r \cdot M$. Following the original suggestion by Shimodaira [158], IQ-TREE generates replicates using $r \in [0.5, \ldots, 1.4]$. For each M_r, a set of bootstrap replicates $\mathcal{B}_r$ with $|\mathcal{B}_r| \geq 10\,000$ is drawn using the RELL method. To make sure that the obtained likelihoods are comparable, the AU test scales them to the same length of the original MSA M using the factor M/M_r. The AU test subsequently applies the same significance test as the SH test described above.

Expected Likelihood Weight Test

Rather than significance testing, Strimmer and Rambaut [171] suggested generating confidence sets using *expected likelihood weights* (ELW). The likelihood weight of a candidate T_x is computed as the fraction of its likelihood L_x over the sum of likelihoods of all candidates $w_x = L_x / \sum_i L_i$. To accept or reject candidates based on the likelihood weight, we need to compute the *expected* likelihood weight under the true model. Since the true model is not known in general, the ELW test estimates the expected likelihood weight based on RELL samples as

$$E[w_x] \approx \bar{w}_x = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} w_x^{(i)},$$

where $w_x^{(i)}$ is the likelihood weight of T_x re-optimized under bootstrap replicate $\boldsymbol{S}^{(i)}$. The likelihood weights sum to 1 across all candidate trees. The ELW test then orders the candidates by the expected likelihood weights and accepts trees with the highest weights while their cumulative sum is below a certain confidence level (typically 0.95). Note that since the ELW test is not a significance test, the posterior weights cannot be interpreted as P-values.

2.1.5 Sequence Simulation

Verifying phylogenetic methods and hypotheses requires knowledge about the true underlying process. Since there only exist but a few known, ground-truth phylogenies, sequence simulation is an important tool for developing novel approaches. In analogy to ML phylogenetic inference, state-of-the-art simulations tools are based on probabilistic models of sequence evolution (Section 2.1.3.2) [28, 114]. While phylogenetic inference reconstructs a phylogeny given the sequence data, simulations generate the sequence data given a phylogeny, thus, essentially reverting the inference process. There are two main approaches to simulate sequences using a given model of sequence evolution: the *probability-matrix* approach and the *rate-matrix* approach [194]. For both approaches, the general idea is to traverse the given input phylogeny top-down until a sequence has been simulated for all inner nodes and leaves. The final, simulated MSA is then simply the matrix comprising the simulated sequences at all leaf nodes. Consequently, we require a *rooted* phylogeny as input for these simulations. Given an unrooted phylogeny, we can determine a root, for instance, at random, or using more elaborate approaches such as *molecular-clock rooting* or *outgroup routing* [192]. Since the models of sequence evolution are time-dependent, both simulation approaches also require branch lengths in the input phylogeny.

Given a rooted phylogeny with branch lengths and a probabilistic model of sequence evolution, the simulation procedure is initialized at the root of the tree. The root sequence is generated by sampling the states of the respective data type based on the equilibrium frequencies $\boldsymbol{\Pi}$ given by the model.

The *probability-matrix* approach generates child sequences based on the transition probability matrix $\boldsymbol{P}(t) = e^{\boldsymbol{Q}t}$ (see Section 2.1.3.2), where t is the branch length of the current branch. A state at a site is simulated by drawing a new state at random from the discrete distribution of possible states with probabilities according to $\boldsymbol{P}(t)$. For instance, if the state of the current node is the nucleotide `A`, the state for the corresponding site in the child sequence is randomly drawn from `A`, `C`, `G`, `T` with probabilities $P_{\texttt{A}\to\texttt{A}}(t)$, $P_{\texttt{A}\to\texttt{C}}(t)$, $P_{\texttt{A}\to\texttt{G}}(t)$, $P_{\texttt{A}\to\texttt{T}}(t)$. Unless the model of sequence evolution accommodates rate heterogeneity, $\boldsymbol{P}(t)$ only needs to be computed once per branch, as it solely depends on the branch length t. For models with discrete rates, $\boldsymbol{P}(t)$ is computed per rate, and for models with a continuous rate heterogeneity distribution, $\boldsymbol{P}(t)$ is computed for each site independently.

In contrast, the *rate-matrix* approach simulates *waiting times* between mutation events and can thus simulate multiple substitution events along a single branch. If the site is currently in state i, the substitution rate of i is $Q_{i\to i} = -\sum_{j\neq i} Q_{i\to j}$. The waiting time until a substitution event occurs is exponentially distributed, with a mean of $1/Q_{i\to i}$. To simulate the state of the same site in the child node, a waiting time s is drawn from this exponential distribution. If the waiting time exceeds the branch length ($s > t$), no substitution occurred between the current node and the child node. Hence, the respective site in the child sequence equals the current state i. If a substitution occurred, the current state is updated by drawing a new state j randomly from the discrete distribution of possible states with probabilities according to $\boldsymbol{P}(s)$ (in analogy to the probability-matrix approach). Since $s \leq t$, the evolutionary time between the current node and its descendant is not reached, and the sampling process is repeated given the new state j and the remaining evolutionary time $t - s$.

In this thesis, we rely on the AliSim sequence simulation tool [114]. AliSim implements a combination of both simulation approaches. The runtime of the rate-matrix approach increases with longer branches, whereas the runtime of the probability-matrix approach is independent of the branch lengths. Thus, AliSim implements an adaptive simulation procedure, that determines the simulation algorithm independently for each branch based on its length. For short branches, the child sequence is simulated using the rate-matrix approach, for long branches AliSim simulates the child sequence using the probability-matrix approach. Ly-Trong *et al.* [114] determined the optimal branch length threshold for this adaptive heuristic based on extensive experiments.

The simulation procedures outlined thus far only simulated substitution events. To simulate indel events, AliSim selects the mutation event (substitution, insertion, deletion) with probabilities S_R, I_R, and D_R, respectively. In case of a substitution event, AliSim simulates the new state according to the procedure described above. In case of an insertion event, AliSim samples an insertion position uniformly at random, and an insertion length from a given insertion length distribution. The inserted sequence is then generated at random based on the equilibrium frequencies. Analogously, in case of a deletion event, AliSim samples a deletion position uniformly at random, and a deletion length from a given deletion length distribution. The

probabilities of each mutation event, as well as the insertion and deletion length distributions, can be estimated from empirical data using dedicated tools, such as SpartaABC [110].

Instead of estimating insertion and deletion parameters to generate indel patterns, AliSim alternatively offers a *mimicking* option. If the underlying (empirical) MSA of the input phylogeny is available, AliSim can superimpose the gap pattern of this MSA onto the simulated MSA, and thereby better mimic the evolutionary history of the empirical MSA.

2.2 Population Genetics

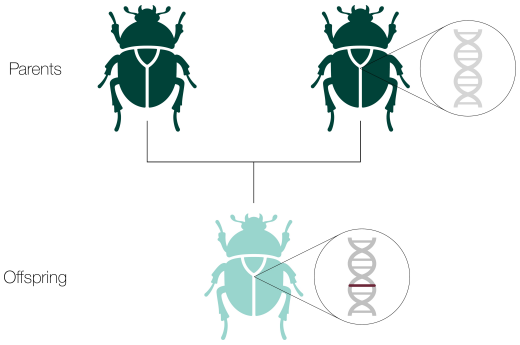
Population genetics studies the genetic composition of populations and its change over time. It focuses on understanding population structure and the underlying evolutionary forces that lead to genetic and phenotypic differences between individuals. Unlike phylogenetics, which primarily studies the evolutionary history *between* distinct species, population genetics focuses on the genetic diversity *within* a species.

The term population structure refers to the genetic variation and distribution of genetic variants within or between closely related populations. Genetic variation is a result of the four evolutionary forces: mutation, gene flow, genetic drift, and natural selection. Figure 2.6 visualizes the effects of these four forces by example of beetles of different colors. *Mutation* is the change of DNA sequences as a result of nucleotide substitutions, insertions, and deletions (Figure 2.2), or as a result of random (re-)combination of the parental DNA during sexual reproduction. For instance, in Figure 2.6(a), the offspring beetle differs in color compared to its parents due to a random mutation of its DNA. *Gene flow* is the result of the migration of individuals between populations, and refers to the transfer of genetic variants from one population to another. In Figure 2.6(b), an individual from the population of brightly colored beetles migrates to the population of darkly colored beetles, leading to a change in its population structure. In human population genetics, this phenomenon is also referred to as *admixture*. *Genetic drift* is the change in composition (frequency of occurrence) of a genetic variant caused by a random event. Figure 2.6(c) visualizes this evolutionary force. In this example, the phenotype distribution changes as a result of the random event of a fire. Finally, the survival and reproduction success probabilities of individuals differ depending on their phenotype. This evolutionary mechanism was termed *natural selection* by Charles Darwin [35] and is visualized in Figure 2.6(d). The crow can spot the bright beetles more easily. Thus, the dark beetle variant has a selection advantage. Understanding population structure can help to study these evolutionary forces.

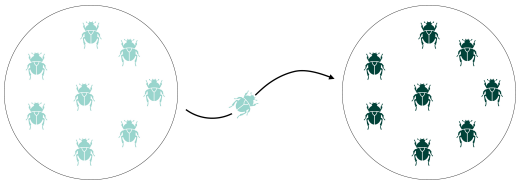
Similar to phylogenetics, population genetics methods rely on molecular genotype or phenotype data. In this thesis, we focus on computational approaches based on genotype data only. Furthermore, we focus on studying population structure using *dimensionality reduction* and *clustering* techniques.

2.2.1 Genotype Data

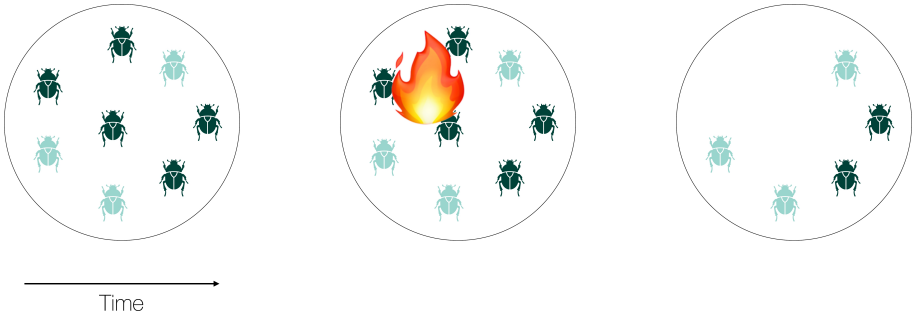
A genetic variant of a specific locus is called *allele*. A locus is the position of a gene or genetic marker along a chromosome. Note that a locus can either describe a single nucleotide or a sequence of nucleotides. *Diploid* organisms have two sets of homologous chromosomes, one maternal and one paternal set. If both sets have the same allele at a specific locus, the organism is *homozygous* in this allele. If they differ, the organism is *heterozygous*. Allele differences within or between individuals can induce phenotypic changes in the respective individual. One of the most famous examples is the study of phenotypic traits of peas by Mendel [117]. For instance, a single gene controls whether the pea seeds are wrinkled or round [19]. Note that



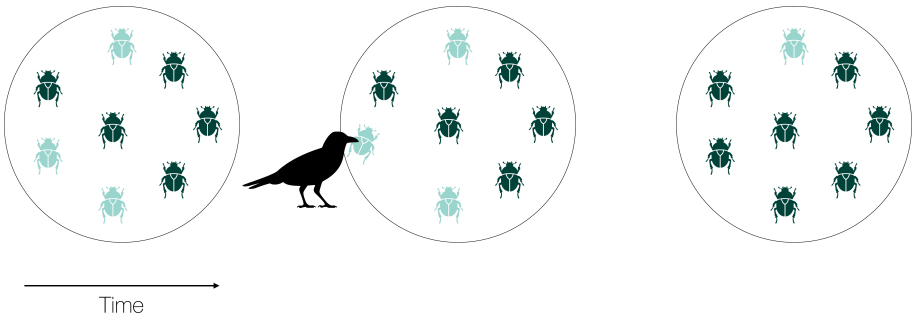
(a) Mutation: The offspring beetle differs in color compared to its parents as a result of a random DNA mutation.



(b) Gene Flow: An individual from the population of brightly colored beetles migrates to a population of darkly colored beetles.



(c) Genetic Drift: The population structure (composition) changed due to a fire that by pure random chance only killed darkly colored beetles.



(d) Natural Selection: A crow can spot brightly colored beetles more easily, leading to a change of color distribution as a result of selective pressure.

Figure 2.6: Exemplary visualization of the four evolutionary forces mutation, gene flow, genetic drift, and natural selection.

not all organisms are diploid, and ploidy can vary between cells and also throughout the life cycle of organisms [185]. In this thesis, we focus on genetic data of diploid organisms.

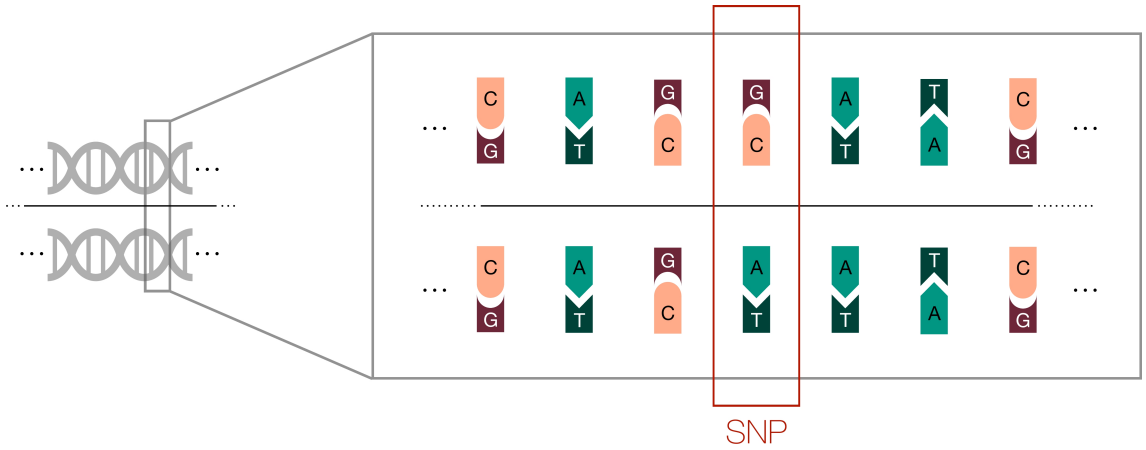


Figure 2.7: Visualization of a SNP. In the selected DNA region, the two strands only differ in a single nucleotide pair: `G` – `C` versus `A` – `T`.

A *Single Nucleotide Polymorphism* (*SNP*, pronounced "snip") is the variation of a single nucleotide among the DNA of individuals of a population. A SNP is associated with a *reference* nucleotide, present in the majority of individuals, and a *variant* nucleotide, present in the minority of individuals. Figure 2.7 visualizes a single SNP at the DNA level of two individuals. The two DNA molecules differ in a single nucleotide pair: `G` – `C` versus `A` – `T`.

A *genotype dataset* (or *genotype data*) comprises sequences of SNPs. For our applications, we assume genotype data to be *biallelic*, meaning that a specific locus in the maternal and paternal genomic data can be either one of two alleles: a or A. Hence, the genotype of an individual has one of the three states: homozygous a (aa), homozygous A (AA), or heterozygous (aA/Aa). To apply mathematical operations to these data, the genotypes need to be encoded via numeric values. One common encoding is the EIGENSTRAT [133] format that encodes the genotypes using the integers 0, 1, and 2, indicating if an individual has zero, one, or two copies of the respective reference allele. A dedicated special character denotes missing genotypes (9 in the case of the EIGENSTRAT format).

The resulting dataset is a matrix

$$\boldsymbol{G} = \begin{bmatrix} g_1^1 & g_1^2 & \dots & g_1^M \\ g_2^1 & g_2^2 & \dots & g_2^M \\ \dots & \dots & \dots & \dots \\ g_N^1 & g_N^2 & \dots & g_N^M \end{bmatrix} \in N \times M, \tag{2.15}$$

where rows correspond to the N individuals (of the same species) under study, and each column corresponds to one of the M SNPs. Typically $M \gg N$; for instance, a dataset of global human genotypes published by Lazaridis *et al.* [102] comprises $M = 605\,775$ SNPs for $N = 2068$ individuals. The genotype matrix is usually accompanied by metadata for the SNPs and the individuals. This metadata

includes, for example, the chromosome number where the SNP is located, or the (sub-)population (for example "Greek") an individual forms part of.

2.2.1.1 Simulating Genotype Data

Simulations are an important tool in population genetics to assess and verify novel analysis techniques, tools, or hypotheses. Simulation approaches can be broadly categorized into three categories [197]. The *coalescent* model simulates sequences starting from currently living samples backward in time until a most recent common ancestor is reached. In contrast, *forward-in-time* models start from an initial ancient population and simulate forward in time to generate subsequent generations of sequences until the current time is reached. Finally, *resampling* approaches simulate sequences based on bootstrap resampling of empirical genomic data.

The coalescent process developed by Kingman [90] simulates a trajectory of individuals backward in time. At certain points in time, single alleles *coalesce*, leading to coalescence of individuals until the most recent common ancestor of all individuals is reached. The probability for a coalescence event to occur in the current generation is modelled as a stochastic process, and depends on the population size. This model can be extended to additionally model recombination and mutation events. Recombination is an exchange of genetic material between individuals or between different loci within a single individual. Mutation refers to the substitution, insertion, or deletion of DNA nucleotides (see Section 2.1.2).

In this thesis, we simulate sequences using the *msprime* [14] simulation tool, which implements a coalescent model. In addition to recombination and mutation events, *msprime* can simulate migration events, that result in an exchange of genetic material between populations, changes in population size, or population structure at various points in time.

Parameterizing simulations to obtain realistic sequences is a challenging task. Thus, in our work, we simulate genomic sequences based on 13 published, empirical human demographic models in the *stdpopsim* catalogue [3, 101]. Each demographic model defines simulation parameters such as the population size, recombination or mutation rates, or migration events for up to 10 populations.

2.2.2 Dimensionality Reduction

Visualizing and analyzing high dimensional genotype data is challenging. Computing and examining two-dimensional plots of all possible combinations of SNPs is not only computationally infeasible as there are $\binom{M}{2}$ possible plots for M SNPs, but it is also unlikely to be informative as the underlying pattern of similarity between individuals might only be visible by combining SNPs. The idea behind dimensionality reduction techniques is to find a representation of the data in a lower dimensional space for easier visualization and interpretation. Popular dimensionality reduction approaches include *Principal Component Analysis* (*PCA*) [134], *Multidimensional Scaling* (*MDS*) [177], *t-distributed stochastic neighbor embedding*

(*t-SNE*) [179], or more recent deep learning-based approaches such as *variational autoencoders* (*VAEs*) [13]. In this thesis, we focus on the PCA and MDS approaches, which we will explain in the following. Note that while the explanation focuses on genotype data, the described approaches translate directly to numeric data matrices in general.

2.2.2.1 Principal Component Analysis

The fundamental idea behind *Principal Component Analysis* (*PCA*) [134] is to find a low-dimensional embedding of the genotype data $\boldsymbol{G}$ that preserves as much information of the higher dimensional data as possible. The underlying assumption is that in the high-dimensional genotype data, structural information is redundant, that is, some SNPs encode similar information as others. With PCA, information is quantified via the data *variance*. The axes of the low-dimensional space are called *principal components*, and each principal component is a linear combination of the M SNPs of the genotype data. PCA is computed such that the first principal component explains the most variance in the data, the second principal component explains the second most variance, and so on. Essentially, PCA performs a high-dimensional rotation of the data such that the axes of highest variance in the original data are the axes of the transformed space, under the restriction that the axes should form an *orthonormal basis*, that is, the axes are orthogonal and of unit length. Note that this does not immediately result in a dimensionality-reduction, but only in a transformation of the data into a space that is now ordered by variance. Only after this transformation, we can reduce the data dimensionality by only using the first L dimensions of this new space. In the following, we explain the mathematical foundation of PCA in more detail. This explanation is based on an excellent summary by Shlens [160]. Note that, while we focus on genotype data and denote the input data using the previously defined matrix $\boldsymbol{G}$, the described method translates to any data matrix $\boldsymbol{X} \in N \times M$ comprising N samples and M features with values in $\mathbb{R}$.

To compute a PCA embedding, the input data needs to be centered such that the mean of the data lies in the origin $\vec{0}$ in all M dimensions. This preprocessing is necessary, as otherwise the first principal component tends to reflect the mean of the data rather than the direction of highest variance. For a more detailed explanation, we refer the interested reader to [122]. In the following, we denote the centered genotype matrix as $\hat{\boldsymbol{G}}$.

Mathematically speaking, PCA corresponds to finding a transformation matrix $\boldsymbol{W}$ of size $M \times M$ such that

$$\boldsymbol{Z} = \hat{\boldsymbol{G}}\boldsymbol{W}, \tag{2.16}$$

where $\boldsymbol{Z}$ is the PCA embedding of $\hat{\boldsymbol{G}}$.

Equation (2.16) describes a change of basis. The columns of $\boldsymbol{W}$ are a set of new basis vectors, and transform the rows of $\hat{\boldsymbol{G}}$ into a new coordinate system. These column vectors are the principal components. In our original coordinate system,

where each axis corresponds to one SNP, the individual SNPs are interrelated. We can express the interrelation of SNPs using the *covariance matrix*. For the centered genotype data $\hat{\boldsymbol{G}}$, the covariance matrix is computed as

$$\boldsymbol{C}_{\hat{G}} = \frac{1}{M}\hat{\boldsymbol{G}}^\top\hat{\boldsymbol{G}}. \tag{2.17}$$

This covariance matrix $\boldsymbol{C}_{\hat{G}}$ is symmetric. Its diagonal entries are the variances within each SNP, and its off-diagonal entries are the covariances between SNPs. The covariance of a SNP pair $\boldsymbol{g}^a$ and $\boldsymbol{g}^b$ describes their relationship. For instance, a positive covariance between $\boldsymbol{g}^a$ and $\boldsymbol{g}^b$ indicates that large values in $\boldsymbol{g}^a$ correspond to large values in $\boldsymbol{g}^b$. If the covariance between $\boldsymbol{g}^a$ and $\boldsymbol{g}^b$ is not zero, the SNPs are correlated, indicating redundancy in the data.

With PCA, the goal is to find a new coordinate system such that the axes are uncorrelated. This means that we want the covariance matrix of the embedding $\boldsymbol{Z}$ to be a *diagonal* matrix. We can compute $\boldsymbol{C}_Z$ as

$$\begin{aligned}
\boldsymbol{C}_Z &= \frac{1}{M}\boldsymbol{Z}^\top\boldsymbol{Z} && (2.18)\\
&\overset{(2.16)}{=} \frac{1}{M}\left(\hat{\boldsymbol{G}}\boldsymbol{W}\right)^\top\left(\hat{\boldsymbol{G}}\boldsymbol{W}\right) && (2.19)\\
&= \frac{1}{M}\boldsymbol{W}^\top\hat{\boldsymbol{G}}^\top\hat{\boldsymbol{G}}\boldsymbol{W} && (2.20)\\
&= \boldsymbol{W}^\top\left(\frac{1}{M}\hat{\boldsymbol{G}}^\top\hat{\boldsymbol{G}}\right)\boldsymbol{W} && (2.21)\\
&\overset{(2.17)}{=} \boldsymbol{W}^\top\boldsymbol{C}_{\hat{G}}\boldsymbol{W} && (2.22)
\end{aligned}$$

Note that in Equation (2.22) we express $\boldsymbol{C}_Z$ using $\boldsymbol{C}_G$. Since $\boldsymbol{C}_{\hat{G}}$ is symmetric, we can diagonalize the matrix via its eigenvectors and eigenvalues

$$\boldsymbol{C}_{\hat{G}} = \boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^\top, \tag{2.23}$$

where $\boldsymbol{E}$ is an $M \times M$ matrix whose i-th column corresponds to the i-th eigenvector of $\boldsymbol{C}_{\hat{G}}$. $\boldsymbol{\Lambda}$ is a diagonal matrix with the respective eigenvalues as diagonal entries. Computing $\boldsymbol{E}$ and $\boldsymbol{\Lambda}$ is called *Eigendecomposition*. By convention, the eigenvalues in $\boldsymbol{\Lambda}$ and their corresponding eigenvectors in $\boldsymbol{E}$ are arranged in descending order of magnitude of the eigenvalues.

If we set $\boldsymbol{W}$ to be the matrix of eigenvectors $\boldsymbol{E}$, $\boldsymbol{C}_Z$ equals the diagonal matrix of eigenvalues $\boldsymbol{\Lambda}$:

$$\boldsymbol{C}_Z \overset{(2.22)}{=} \boldsymbol{W}^\top\boldsymbol{C}_{\hat{G}}\boldsymbol{W} \tag{2.24}$$

$$\overset{(2.23)}{=} \boldsymbol{W}^\top \left(\boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^\top\right)\boldsymbol{W} \tag{2.25}$$

$$\overset{\boldsymbol{W}:=\boldsymbol{E}}{=} \boldsymbol{W}^\top \left(\boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^\top\right)\boldsymbol{W} \tag{2.26}$$

$$= \boldsymbol{\Lambda}. \tag{2.27}$$

Thus, the principal components in $\boldsymbol{W}$ are the eigenvectors of the covariance matrix $\boldsymbol{C}_{\hat{G}}$. The eigenvalues in $\boldsymbol{\Lambda}$ correspond to the variance per dimension of the PCA embedding space. The sum over all eigenvalues yields the overall amount of variance in the genotype dataset. Dividing each eigenvalue by this sum thus corresponds to the proportion of variance explained by the respective eigenvectors, and thus by the respective principal component.

Computing the matrix product $\hat{\boldsymbol{G}}^\top\hat{\boldsymbol{G}}$ in Equation (2.17) is computationally intense, with a complexity of $\mathcal{O}(M^2N)$. When using *Singular Value Decomposition* (*SVD*), the embedding can be computed directly on the data $\hat{\boldsymbol{G}}$. With SVD, we can factorize $\hat{\boldsymbol{G}}$ such that

$$\hat{\boldsymbol{G}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top, \tag{2.28}$$

where $\boldsymbol{W}$ corresponds to the eigenvectors of $\hat{\boldsymbol{G}}^\top\hat{\boldsymbol{G}}$. Note that this is proportional to the covariance matrix $\boldsymbol{C}_{\hat{G}}$ by a factor of $1/M$. $\boldsymbol{\Sigma}$ is a diagonal matrix of the singular values and equals the square root of $\boldsymbol{\Lambda}$.

Using this factorization, we can reformulate the computation of the PCA embedding $\boldsymbol{Z}$ as

$$\boldsymbol{Z} \overset{(2.16)}{=} \hat{\boldsymbol{G}}\boldsymbol{W} \tag{2.29}$$

$$= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top\boldsymbol{W} \tag{2.30}$$

$$= \boldsymbol{U}\boldsymbol{\Sigma}. \tag{2.31}$$

Finally, with PCA, we aim to reduce the dimensionality of the data. Since the eigenvalues in $\boldsymbol{\Lambda}$ (and analogously the singular values in $\boldsymbol{\Sigma}$) are ordered by decreasing magnitude, we can reduce the dimensionality if we only keep the first L principal components, thus defining a lower-dimensional embedding as

$$\boldsymbol{Z}_L = \hat{\boldsymbol{G}}\boldsymbol{W}_L \tag{2.32}$$

$$= \boldsymbol{U}_L\boldsymbol{\Sigma}_L. \tag{2.33}$$

The proportion of variance of the genotype data explained by this lower-dimensional embedding is simply the sum over the proportion of explained variance per principal component for the first L principal components.

Figure 2.8 visualizes the outlined PCA procedure. In this example, the input data is two-dimensional and comprises 500 individuals. The left plot shows the input

Figure 2.8: Example of a PCA embedding of two-dimensional input data. The left figure plots the input data, and the two principal components that we identified using PCA. The right figure shows the transformed data, that is, the PCA embedding of the exemplary input data. The first principal component explains 85% of the data variance, and the second principal components explains the remaining 15% of variance.

data. We can clearly see that the two dimensions are correlated. Using PCA, we can identify the two axes of highest variation, that is, the principal components. The two dark arrows shown in the left plot visualize these axes, with PC 1 corresponding to the axis of highest data variation. Once we have computed the principal components (PC 1 and PC 2), we transform the input data using the transformation matrix $\boldsymbol{W} = [\boldsymbol{PC}\ \mathbf{1}, \boldsymbol{PC}\ \mathbf{2}]$. This essentially results in a rotation of the space such that the principal components form the axes of the new PCA embedding space. The right plot visualizes the embedding of the input data after this transformation.

PCA requires a *complete* dataset, that is, a dataset without missing values, which is generally not the case for genotype data. Consequently, we need to handle missing data before PCA analyses. One straight-forward solution is to remove individuals or SNPs that contain missing values. However, removing SNPs generally reduces the information content, and thus the signal of the underlying population structure in the data. Removing individuals is only possible if the removed individuals are not the core focus of the study. Commonly, instead of removing missing data, missing values are *imputed*. Imputation means that we replace a missing value with an estimate, for instance, the average genotype of the respective SNP (*mean imputation*).

In this thesis, we perform PCA analyses on genotype data using the implementations in the Python machine learning library *scikit-learn* [135] and in *smartpca* [133]. Both implementations include the centering of the genotype data and compute the principal components using SVD. The *smartpca* implementation additionally includes mean imputation.

2.2.2.2 Multidimensional Scaling

In contrast to PCA, *Multidimensional Scaling* (*MDS*) [177] does not reduce the dimensionality of the genotype matrix directly. Instead, MDS relies on a matrix of pairwise distances $\boldsymbol{D} \in N \times N$ between N individuals. The goal of MDS is to find a configuration of individuals in an M-dimensional space that best explains the given distance matrix. Only subsequently, the dimensionality of the data is reduced by using the first L dimensions of the reconstructed data only. The advantage of this approach is that the similarity between individuals can be expressed using any distance metric. In population genetics, we compute the required distance matrix $\boldsymbol{D}$ using the genotype data $\boldsymbol{G}$.

We first motivate and derive the *classical* MDS algorithm that assumes Euclidean distances in $\boldsymbol{D}$. This explanation is based on Mead [116]. Afterward, we discuss how MDS can be applied to non-Euclidean distances.

Given the assumption that $\boldsymbol{D}$ contains Euclidean distances, the distance matrix is computed as

$$\boldsymbol{D} = (d_{ij}) = \|\boldsymbol{g}_i - \boldsymbol{g}_j\|_2, \tag{2.34}$$

where $\boldsymbol{g}_i$ denotes the row vector of all M SNPs recorded for the i-th individual in $\boldsymbol{G}$. Instead of directly operating on this distance matrix, MDS operates on the matrix $\boldsymbol{D}^{(2)}$ of squared distances

$$\boldsymbol{D}^{(2)} = d_{ij}^{(2)} = d_{ij}^2 = \|\boldsymbol{g}_i - \boldsymbol{g}_j\|_2^2. \tag{2.35}$$

The motivation for using squared distances becomes evident in the following, when we derive the mathematical solution to reconstruct $\boldsymbol{G}$. The Euclidean distance is translation invariant. This means that if we shift each individual genotype data $\boldsymbol{G}$ by a factor of c, the distances in $\boldsymbol{D}$ will remain the same. To account for this translation invariance, MDS centers $\boldsymbol{D}$ such that its centroid is in the origin $\vec{0}$. We denote the centered matrix as $\hat{\boldsymbol{D}}$.

The centered distance matrix is the *Gram* matrix of the inner products of $\boldsymbol{G}$

$$\hat{\boldsymbol{D}} = \boldsymbol{G}\boldsymbol{G}^\top, \tag{2.36}$$

which is a symmetric $N \times N$ matrix. Using Eigendecomposition, we can factorize $\hat{\boldsymbol{D}}$ such that

$$\hat{\boldsymbol{D}} = \boldsymbol{G}\boldsymbol{G}^\top = \boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^\top, \tag{2.37}$$

where $\boldsymbol{E}$ denotes the matrix of eigenvectors of $\hat{\boldsymbol{D}}$ and $\boldsymbol{\Lambda}$ the diagonal matrix of the respective eigenvalues. Thus, we can compute the genotype matrix $\boldsymbol{G}$ as

$$\boldsymbol{G} \overset{(2.37)}{=} \boldsymbol{E}\sqrt{\boldsymbol{\Lambda}}, \tag{2.38}$$

based on the Eigendecomposition of the centered distance matrix $\hat{\boldsymbol{D}}$. Using only the first L eigenvectors and eigenvalues defines an L-dimensional embedding $\boldsymbol{Z}_L$ of $\boldsymbol{G}$.

This closed-loop solution finds a perfect configuration of $\boldsymbol{G}$ in an L-dimensional space if the pairwise distances in $\boldsymbol{D}$ are Euclidean. In this case, the L-dimensional embedding using MDS is identical to the L-dimensional embedding computed using PCA [73].

If the pairwise distances in $\boldsymbol{D}$ are non-Euclidean, this outlined approach cannot necessarily reconstruct the original genotype data $\boldsymbol{G}$ exactly. Instead, MDS will find a configuration $\hat{\boldsymbol{G}}$ of N individuals in an L dimensional space such that

$$d_{ij} \approx \delta_{ij} = \|\hat{\boldsymbol{g}}_i - \hat{\boldsymbol{g}}_j\|_2. \tag{2.39}$$

In this thesis, we use the implementation of classical MDS in *scikit-allel* [120].

In addition to pairwise distances between individuals, distances between groups of individuals (or populations) are also commonly used in population genetics studies. For instance, the F_{ST} population distance reflects the proportion of total heterozygosity that can be explained by the within-population heterozygosity [190]. The MDS algorithm outlined above works the same when using per-population instead of per-individual distances. However, the size of the distance matrix is reduced from $N \times N$ to $P \times P$, with P being the number of distinct (sub-)populations in the genotype data. Consequently, the result of MDS will not be a genotype matrix comprising N individuals, but a matrix comprising P samples, where each sample corresponds to one population.

In analogy to PCA, MDS also requires a complete distance matrix and cannot handle missing data. In our application, we compute the distance based on the genotype data $\boldsymbol{G}$ and subsequently reduce the dimensionality via MDS. Thus, the distance matrix in our setup never contains missing data. However, to compute the distance matrix $\boldsymbol{D}$, we need to impute missing values in $\boldsymbol{G}$, for instance, via mean imputation.

2.2.3 Clustering

Clustering methods group individuals based on their similarity, and are useful tools for detecting structure in data. In population genetics, clustering can help to identify genetic (sub-)populations [148], or study the ancestry of individuals [165]. Usually, instead of relying on the high-dimensional genotype data $\boldsymbol{G}$, clustering is applied to a lower-dimensional embedding $\boldsymbol{Z}_L$ of $\boldsymbol{G}$ obtained via PCA or MDS.

While there exists numerous distinct clustering approaches, we focus on k-means clustering in this thesis. K-means clustering assigns the N individuals in $\boldsymbol{Z}_L$ to k disjoint clusters. The k-means problem is $\mathcal{NP}$-hard [6]. However, we can formulate k-means clustering as an optimization problem that tries to minimize the *within-cluster sum of squares* (*WCSS*)

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \boldsymbol{K}_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2, \tag{2.40}$$

where $\boldsymbol{K}_i$ is the i-th cluster and $\boldsymbol{\mu}_i$ the respective cluster centroid

$$\boldsymbol{\mu}_i = \frac{1}{|\boldsymbol{K}_i|} \sum_{\boldsymbol{x} \in \boldsymbol{K}_i} \boldsymbol{x}. \tag{2.41}$$

The WCSS essentially measures how coherent the individual clusters are. The most commonly used heuristic to solve this optimization problem is *Lloyd's algorithm* [109]. Lloyd's algorithm initializes the k cluster centers using k randomly selected individuals from $\boldsymbol{Z}_L$. Subsequently, it alternates between two steps. First, assign each of the N individuals to the cluster with the closest center according to the Euclidean distance. Then, update the cluster centers using the assigned individuals according to Equation (2.41). The cluster assignments and recomputations of centroids are repeated until the cluster assignments remain constant, or until a convergence criterion is reached, for instance, a maximum number of iterations. This heuristic does not necessarily find the optimal solution and heavily depends on the initialization of the cluster centers [10]. Thus, instead of running Lloyd's algorithm once, we usually repeat the process multiple times with different random initializations and select the result with the lowest WCSS.

In this thesis, we use k-means clustering as implemented in *scikit-learn* [135], which relies on Lloyd's algorithm. Instead of a random initialization of cluster centers, the scikit-learn implementation uses a variant of *k-means++* [10] as initialization method. With this approach, the first cluster center is chosen at random from the N individuals in $\boldsymbol{Z}_L$. The second cluster center is based on a weighted random sample of the remaining $N - 1$ individuals. The weights are assigned according to the distance of each individual to the first cluster center, such that the point with the highest Euclidean distance is most likely to be chosen as the second center. This process is repeated until all k centers have been initialized.

2.3 Machine Learning

Machine learning is a field of artificial intelligence that uses statistical algorithms to detect patterns in data and generalize these learned patterns to unseen data. Machine learning algorithms can be broadly categorized into *supervised* and *unsupervised* approaches [84]. In a supervised setting, each training datum is annotated by its target value (*label*), and the machine learning model leverages these labels during training. In contrast, unsupervised learning approaches detect patterns or structures in *unlabeled* data, that is, data without a target value associated with each datum. Clustering approaches, or dimensionality reduction techniques that were discussed in the previous section, are examples of unsupervised learning methods. For the remainder of this section, we will focus on supervised learning.

Machine learning tasks can be separated into *classification* and *regression* tasks [84]. In classification, the target is to categorize data into a pre-defined set of *categories* (or *classes*). A typical example is spam classification for emails. Regression tasks, on the other hand, predict continuous values, for example, the price of a house.

In this thesis, we further differentiate between *classical machine learning* and *deep learning* approaches. Classical machine learning techniques, such as linear regression or Decision Trees, deploy simpler statistical models, while deep learning methods rely on neural networks to automatically learn complex patterns. Classical machine learning models require (manual) feature engineering. However, explaining model decisions and learning processes, for example via feature importance, is straightforward. In contrast, deep learning approaches require only minimal data preprocessing as they automatically learn important features during training, but, interpreting these features requires (extensive) additional analyses [107].

In the following, we outline the training procedure of machine learning models, and discuss evaluation metrics for classification and regression tasks. We further explain selected learning algorithms in greater detail. Finally, we briefly summarize two explainability approaches for classical machine learning models.

2.3.1 Notation

We first define some relevant notation for the following section. A supervised machine learning model is trained on a *training dataset* $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ where $\boldsymbol{x}_i$ denotes the training datum and y_i the respective prediction target called (ground-truth) *label*. For classical machine learning approaches, we rely on p manually engineered *prediction features*. In this case, $\boldsymbol{x}_i = (x_i^1, x_i^2, \dots, x_i^p)$ is the vector of feature values for all p prediction features for the i-th training datum. The *target vector* $\boldsymbol{y} = (y_1, y_2, \dots, y_n)^\top$ is the column vector of all labels, and the *feature matrix* (or *design matrix*) $\boldsymbol{X}$ is a matrix of row vectors $\boldsymbol{x}_i$. For classical machine learning models with pre-defined features, the dimension of $\boldsymbol{X}$ is $n \times p$. We denote a machine learning model as f and the trainable parameters as $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots)$. The trainable parameters are the variables within a machine learning model that are adjusted during training to optimize the fit to the training data. We further denote the prediction of model f for a datum $\boldsymbol{x}_i$ as $\hat{y}_i = f(\boldsymbol{x}_i)$, and the column vector comprising

all predictions as $\hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n)^\top$. Training a machine learning model is also referred to as *fitting the model to the data*. Throughout this thesis, we refer to a trained machine learning model as *predictor* for regression tasks and as *classifier* for classification tasks.

2.3.2 Training

Training a machine learning model requires optimizing its parameters to minimize the prediction error. This prediction error is quantified after an *epoch*, that is, a single complete iteration over each training sample $\boldsymbol{x}_i$. The error is quantified via a *loss function* $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ which reflects how well the predictions $\hat{\boldsymbol{y}}$ approximate the ground-truth labels $\boldsymbol{y}$. The loss function guides the learning process of the prediction model as it is used to update the model's parameters $\boldsymbol{\beta}$ to optimize the fit to the data. One common iterative optimization technique is *gradient descent* [149]. Model parameters are updated after each epoch in the direction of the steepest descent, as determined by the gradient of the loss function. Consequently, a loss function for training needs to be differentiable. A commonly used loss function for regression tasks is the *mean squared error* (MSE)

$$L_{\text{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.42}$$

For binary classification tasks, the *binary cross-entropy* (BCE)

$$L_{\text{BCE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{2.43}$$

is a commonly used loss function.

To evaluate the performance of a trained model and to ensure that it generalizes well to unseen data, we split the full dataset into two disjoint sets, the *training* and the *test dataset*. The model is trained only on the training dataset, and the test dataset is exclusively used to subsequently evaluate the trained model. Typically, the training dataset contains more samples than the test dataset to leverage as much data as possible for training the model, for instance using 80% of the data for training and 20% for performance evaluation. What proportion of data to use in the training dataset depends, among other factors, on the total amount of data and the complexity of the machine learning task [72]. Using a train-test-split can lead to a highly variable performance on the test set depending on what samples are included in the train and test set respectively. An alternative approach to training a model is k-fold cross validation. In k-fold cross validation, the full dataset is divided into k equally sized subsets and the model is trained using $k-1$ subsets for training and the remaining subset for testing. The training procedure is repeated k times such that each subset is used for testing exactly once. The overall model performance is then reported using a summary statistic over all k iterations, for example the average. In practice, $k := 5$ or $k := 10$ are common settings as a tradeoff between computational overhead and performance variance [84].

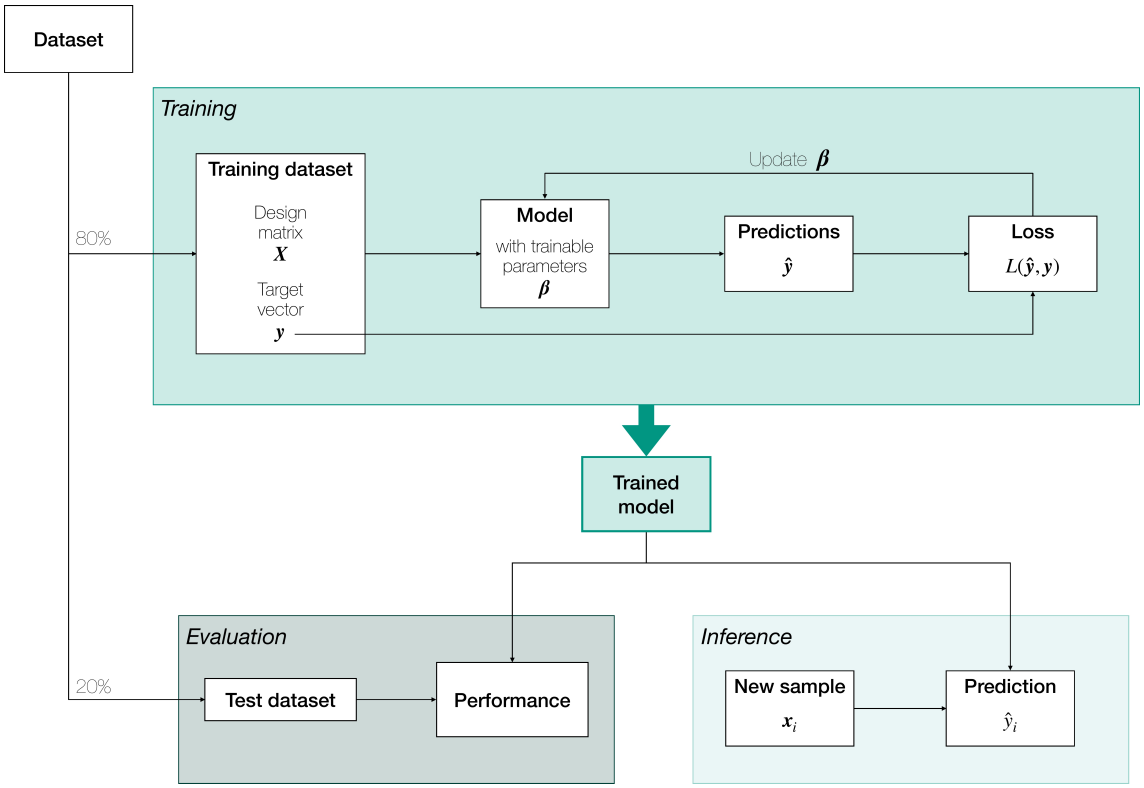


Figure 2.9: High-level overview of the training, evaluation, and inference of a machine learning model using an 80/20 train-test-split.

Figure 2.9 shows a high-level overview of this training and evaluation procedure. It further depicts the application of the trained model for *inference*, that is, for predicting the target $\hat{y}_i$ for a new sample $\boldsymbol{x}_i$. The depicted pipeline uses an 80/20 train-test-split.

Hyperparameters are configuration settings that control the training procedure of machine learning models, for example the depth of a Decision Tree or the learning rate in a Gradient Boosted Tree model (see Section 2.3.4.2). Unlike the model parameters $\boldsymbol{\beta}$ that are learned during training, the hyperparameters need to be defined before starting the training process. These hyperparameters need to be carefully optimized for the task at hand, as they are crucial for obtaining an effective model [191]. Hyperparameter optimization is the process of systematically exploring the hyperparameter space. Optimization techniques range from simple methods like grid search, which tests all possible combinations on a predefined grid, to more sophisticated methods such as Bayesian optimization [124]. Bayesian optimization builds a probabilistic model of the objective function (similar to a loss function) and subsequently selects the most promising hyperparameter combinations based on this probabilistic model. In this thesis, we use the Bayesian optimization strategy as implemented in the *Optuna* framework [4].

For classical machine learning methods, we need to manually define prediction features. How many features we engineer constitutes a trade-off between predictive

power, that is, how well the set of features approximates the underlying pattern, and the size of the training dataset. In general, the more features we provide to a model, the more training data we will require. Adding dimensions to the feature matrix increases the volume of the feature space exponentially. This induces a sparsely sampled feature space if our training dataset is small compared to the number of prediction features. This phenomenon is referred to as the *curse of dimensionality* [84]. Additionally, the more features we include, the higher the risk of *collinearity* (individual features being correlated) and *multicollinearity* (combinations of features being correlated). Correlated features can lead to reduced prediction performance and can yield erroneous estimates of feature importance [84]. While classical machine learning models tend to have few trainable parameters, they can be trained using only hundreds or thousands of training samples (given a small number of prediction features). In contrast, training deep neural networks requires millions or billions of training samples due to the vast number of parameters. For example, the popular large language model *GPT-3* has 175 *billion* trainable parameters and was trained on a dataset of nearly a *trillion* words [26].

The training process needs to be carefully monitored to prevent *overfitting* the training dataset. Overfitting is a result of fitting the model to noise in the data rather than to the true underlying pattern. The training error under such a scenario is low, but the error on the test dataset is high, as the learned noise pattern does not generalize to unseen data [72]. In contrast to overfitting, with *underfitting* a model fails to capture the underlying pattern in the data. In this case, the training performance and the test performance are low. Overfitting and underfitting can be controlled via careful model selection and model parameter tuning. One aspect of model selection is the *bias-variance trade-off* [72]. Bias and variance are error types that affect model performance. Bias refers to the error induced by approximating a (potentially complex) problem using a model that is too simple to capture the complexity of the problem. High bias can lead to underfitting. Variance describes the sensitivity of models to changes in the training data. For models with high variance, small changes in the composition of the training data induce large changes in the parameters of the trained model. High variance can cause overfitting. As a model's complexity, and thus flexibility to fit more complex problems, increases, the variance increases and the bias decreases. While more complex models are more prone to overfitting, we can restrain overfitting via *regularization* techniques [84]. The idea of regularization is to penalize complex models by adding a *regularization term* $R(f)$ to the loss function during training

$$L'(\hat{\boldsymbol{y}}, \boldsymbol{y}) = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \xi R(f). \tag{2.44}$$

The weight parameter ξ controls the importance of the penalty and is a tunable hyperparameter. The most common methods in classical machine learning are L1 and L2 regularization [84]. With L1 regularization, the regularization term $R(f)$ is the L1 norm of the model's parameters

$$R_{\mathrm{L1}} = \|\boldsymbol{\beta}\|_1 = \sum_j |\beta_j|. \tag{2.45}$$

As a result, L1 regularization sets the parameters for the least important features to 0. It thus selects the most important features and thereby restricts the parameter space. In contrast, L2 penalizes large absolute parameter values using the L2 norm as the regularization term

$$R_{\mathrm{L2}} = \|\boldsymbol{\beta}\|_2^2 = \sum_j \beta_j^2. \tag{2.46}$$

Penalizing large parameters prevents the model from heavily relying on a few features only.

2.3.3 Performance Evaluation

Using the independent test dataset, we can evaluate the performance of a trained model. In contrast to the loss function for training the model, a metric for performance evaluation does not need to be differentiable. Evaluation metrics are model-agnostic. This means that they are independent of the internal structure of the chosen learning model, as they only compare the prediction $\hat{\boldsymbol{y}}$ to the ground-truth labels $\boldsymbol{y}$. Hence, we can compare different trained models for the same task by comparing their performance via appropriate evaluation metrics. The choice of the evaluation metric depends on the task at hand. In the following, we present selected evaluation metrics for classification and regression tasks. Note that the applications we discuss in this thesis are binary classification and regression tasks, and we therefore focus on respective metrics.

Classification

Many (binary) classification evaluation metrics are based on the so-called *confusion matrix*, that visualizes the predictive performance of a classifier:

		Predicted	
		Positive	Negative
Actual	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

The sum over all four cells (TP+FN+FP+TN) equals the total number of samples. The sum of the diagonal entries (TP + TN) corresponds to the number of correct predictions. Based on this confusion matrix, we can define the *Accuracy* (ACC) of a classifier as the proportion of correct predictions over all samples:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP} + \mathrm{TN}}. \tag{2.47}$$

The ACC is easy to interpret, but it can be misleading in the case of imbalanced datasets. The *Balanced Accuracy* (BACC) is better suited for imbalanced datasets. The BACC is the average of the *true positive rate* (TPR, also called *sensitivity*) and the *true negative rate* (TNR, also called *specificity*)

$$\text{BACC} = \frac{1}{2}\left(\text{TPR} + \text{TNR}\right) = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP}+\text{FN}} + \frac{\text{TN}}{\text{FP}+\text{TN}}\right). \tag{2.48}$$

The TPR quantifies how many of the actual positive samples the classifier correctly predicted as being positive. Analogously, the TNR quantifies the fraction of actual negative samples that were correctly predicted as being negative.

Regression

The prediction target for regression tasks is a continuous, numeric value. Popular metrics for evaluating trained regression models include the R^2, the *Mean Squared Error* (MSE), the *Mean Absolute Error* (MAE), and the *Mean Absolute Percentage Error* (MAPE).

The R^2 metric quantifies the proportion of variance explained by the model. R^2 is computed as

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{2.49}$$

where the RSS (*residual sum of squares*) is the sum of the squared prediction errors, and the TSS (*total sum of squares*) measures the variance of the target vector $\boldsymbol{y}$. $\bar{y}$ denotes the average prediction target.

The R^2 assumes a value between 0 and 1, with higher values indicating a better model fit. This metric is easy to interpret. However, what constitutes a *good* R^2 is highly context-dependent, and it is unreasonable to assume the existence of a model with $R^2 = 1$ for all machine learning tasks [84]. Moreover, R^2 assumes that a *linear* relationship between the data $\boldsymbol{X}$ and the target $\boldsymbol{y}$ exists.

A more commonly used measure is the *mean squared error* (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{2.50}$$

The MSE is smaller the better the predictions approximate the ground-truth labels. However, since the MSE penalizes large errors, it is sensitive to outliers.

The *mean absolute error* (MAE) is the average magnitude of prediction errors

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{2.51}$$

The interpretation of the MAE is straightforward, with lower values indicating a better model fit.

The *mean absolute percentage error* (MAPE) quantifies the prediction error as a percentage of the ground-truth labels, thus indicating the relative size of errors

$$\text{MAPE} = 100 \cdot \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|. \tag{2.52}$$

The MAPE, while being easy to interpret, is undefined for labels $y_i = 0$. Many implementations of the MAPE add a small constant ε to the denominator to prevent a division by 0. However, this makes the MAPE disproportionately large for values close to 0.

Each of these metrics covers a distinct aspect of model performance evaluation that helps to guide model selection and refinement. Due to their individual constraints and drawbacks, it is advantageous to use a combination of multiple metrics.

2.3.4 Learning Algorithms

Over the course of the past decades, a plethora of machine learning algorithms have been developed, ranging from simple models such as *linear regression* to complex models such as (deep) neural networks. With the most recent developments in artificial intelligence research, complex deep neural networks with advanced model architectures such as *transformer* networks [180] have become available, demonstrating high performance on various tasks [106]. Choosing the learning algorithm that is best suited for the task at hand is challenging. The most obvious selection criterion is model performance on an independent test dataset. However, aspects such as the bias-variance-tradeoff, the availability of training data, model explainability, or the potentially time-consuming and complex process of feature engineering need to be considered as well.

In this thesis, we rely on *tree-based* models for regression and classification tasks, as well as *logistic regression* and *Convolutional Neural Networks* for classification tasks. In the following, we describe these approaches in greater detail. Our reasoning for the choice of models is a combination of explainability, availability of training data, and computational complexity of ground-truth label computation. We provide further reasoning in the respective chapters.

2.3.4.1 Linear and Logistic Regression

The simplest classical machine learning approaches are *linear regression* for regression tasks and *logistic regression* for classification tasks. Both approaches are *generalized linear approaches*, that is, the model's output is a linear combination of the prediction features [84].

Linear regression predicts the quantitative response $\hat{y}_i$ of a datum $\boldsymbol{x}_i$ as the weighted sum over the respective features values

$$\hat{y}_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_i^j + \epsilon_i = \beta_0 + \beta_1 x_i^1 + \cdots + \beta_p x_i^p + \epsilon_i, \tag{2.53}$$

where ϵ_i is the *error term* or *noise* of the model that captures the *unmodelled* relationship between the target y_i and the features values $\boldsymbol{x}_i$. Unmodelled means that there is an aspect of the problem that is not represented by a feature. The model parameters $\boldsymbol{\beta}$ are learned during training by maximizing the model's fit to the data. This can be achieved by minimizing the error term $\boldsymbol{\epsilon} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{\epsilon}$ is the vector of error terms ϵ_i and $\boldsymbol{\beta}$ is the $p+1$-dimensional vector of model parameters. Usually, instead of minimizing $\boldsymbol{\epsilon}$ directly, the respective sum of squared errors $\|\boldsymbol{\epsilon}\|_2^2$ is minimized. Common methods to find good parameters $\boldsymbol{\beta}$ include *least-squares* or *maximum likelihood* estimation.

The prediction output of this linear regression model is *quantitative*. In a classification setting, the target value is *qualitative*, since we intend to categorize data into pre-defined categories. Ideally, we would want to predict the probability $P(C_j|\boldsymbol{x}_i)$ that the datum $\boldsymbol{x}_i$ belongs to a certain category C_j. Using linear regression for classification poses two challenges. First, encoding qualitative targets into quantitative values is not straightforward, as any numbering would suggest a natural order of, or a distance between, categories. This is less problematic under binary classification settings with categories C_0 and C_1. In such a setting, we could, for example, predict C_1 if $\hat{y}_i > 0.5$. However, linear regression can predict values outside the $[0, 1]$ interval and these values do not provide reasonable probability estimates [84]. Instead, we use a variant of linear regression called *logistic regression*. Note that logistic regression usually refers to a model for binary classification. An extension of this model to more than two categories is referred to as *multinomial logistic regression* [84].

For binary logistic regression with categories C_0 and C_1, the output of linear regression is transformed to a $[0, 1]$-interval using the *logistic function*

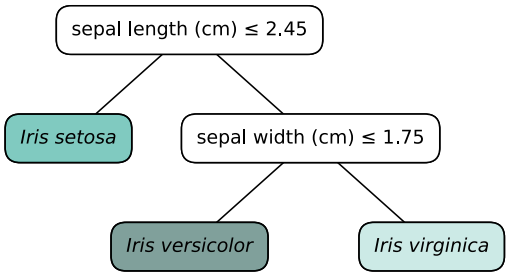
$$P(C_j|\boldsymbol{x}_i) = \frac{e^{\beta_0 + \sum_{j=1}^{p} \beta_j x_i^j}}{1 + e^{\beta_0 + \sum_{j=1}^{p} \beta_j x_i^j}}. \tag{2.54}$$

Similar to linear regression, we can fit a logistic regression model to the data using maximum likelihood estimation.

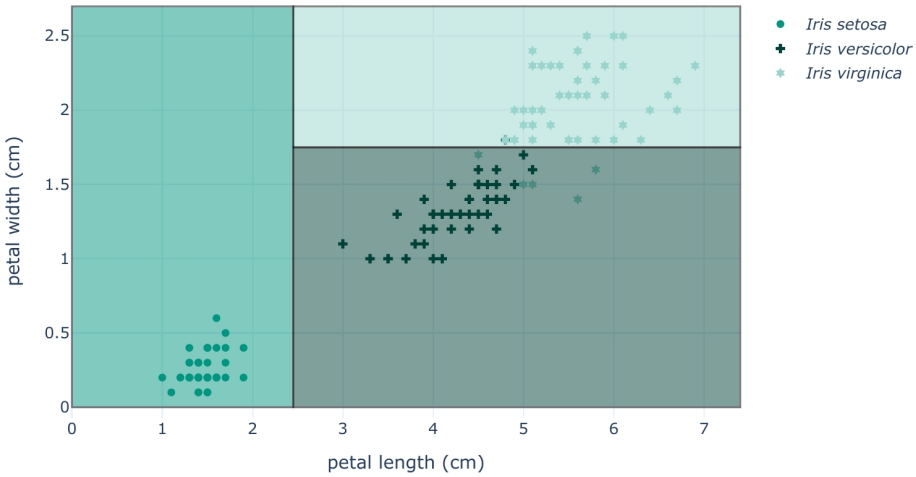
2.3.4.2 Tree-Based Models

Tree-based models are based on *Decision Trees*. A Decision Tree recursively segments the feature space into multiple *decision regions*. The prediction for a certain sample corresponds to the decision region to which it belongs. The respective split decisions leading to these regions are learned during training and can be visualized as a tree with inner nodes representing decisions (splits), and leaves representing the learned decision regions.

In the following, we explain and visualize the underlying concept of Decision Trees using a popular toy dataset for classification. The *Iris* dataset [8, 51] contains 150 samples with three target categories (*Iris setosa*, *Iris virginica*, and *Iris versicolor*). Each sample has four morphological features: the *sepal* length and width and the *petal* length and width.



(a) Decision tree structure.



(b) Decision regions. Black horizontal and vertical lines correspond to split decisions. Each region corresponds to one leaf in the Decision Tree, and colors indicate the predicted category.

Figure 2.10: Decision tree and the respective decision regions for a tree trained on the *Iris dataset* with a maximum depth of 2 using the petal width and petal length as prediction features.

A Decision Tree is trained recursively and top-down. For each split, the goal is to find the feature and respective value that best separates the current set of training samples according to a pre-defined criterion. Common split criteria for classification tasks include the *Gini impurity* or the *Entropy*, that both measure the variance across categories. The MSE or MAE are common split criteria for regression tasks. Figure 2.10(a) shows a Decision Tree for classifying samples of the *Iris* dataset using the petal length and petal width as prediction features. Figure 2.10(b) shows the respective decision regions.

Ultimately, this splitting process will lead to a single leaf for each training datum and will consequently overfit the training data. To prevent overfitting when training a Decision Tree, we can either configure early-stopping criteria, or build a full tree and subsequently prune it by (iteratively) merging leaf nodes. Typical criteria for both approaches are a maximum tree depth or a maximum leaf number [84]. These

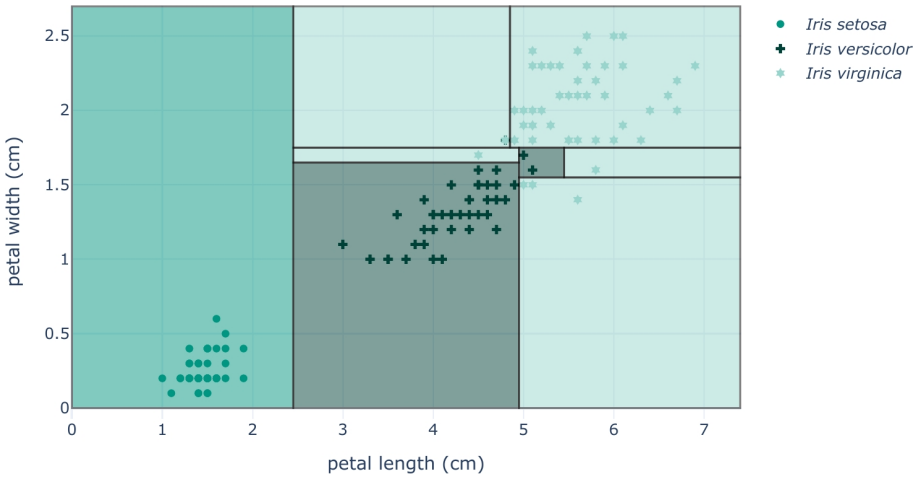


Figure 2.11: Decision regions of a Decision Tree trained on the *Iris dataset* without a depth constraint. Black horizontal and vertical lines correspond to split decisions. Each region corresponds to one leaf in the Decision Tree, and colors indicate the predicted category.

hyperparameters of the tree are dataset-dependent and should be optimized for the task at hand.

We constrained the Decision Tree in Figure 2.10(a) to a maximum depth of 2 to prevent overfitting. For the decision regions visualized in Figure 2.11, we removed this constraint. Compared to the decision regions in Figure 2.10(b), only a single *Iris versicolor* training datum is misclassified, simply because it has the same petal length and width as an *Iris virginica* training datum. However, the respective Decision Tree is substantially more complex with a depth of 5 and comprises 8 leaf nodes.

Considering these fine-grained decision regions, it is clear that a single Decision Tree has high variance and is susceptible to small changes in the training data. This has been shown to be a general problem of stand-alone Decision Trees [84]. We can combine multiple Decision Trees into a single predictor to improve accuracy and robustness using the *Bagging* and *Boosting ensemble* methods [84]. *Bagging* is short for *bootstrap aggregation* and relies on Decision Trees trained on bootstrapped training datasets. Such bootstrapped training datasets are obtained by repeatedly resampling the training data with replacement. Each individual Decision Tree is trained on a separate bootstrapped training dataset. A *Random Forest* is a learning algorithm that relies on bagging [23]. With *boosting*, we train Decision Trees sequentially. Compared to bagging, boosting does not rely on bootstrapped datasets, but instead uses a *modified* version of the training dataset for training each individual Decision Tree. Boosting algorithms mainly differ in the implemented modification of the dataset. For instance, *AdaBoost* [58] weights the training samples according to their current prediction error to grow the current Decision Tree in favor of samples with a large error. In this thesis, we rely on the *Gradient Boosted Trees* (GBT) algorithm [59]. The respective underlying boosting algorithm modifies the training

data in each iteration by modifying the prediction target instead of weighting the individual training samples.

Note that ensemble approaches are applicable to other statistical learning methods as well. Yet, here, we only focus on ensembles in the context of Decision Trees.

Random Forest

A *Random Forest* [23] relies on bagging and combines multiple Decision Trees that are individually trained on bootstrapped datasets. The final prediction of this ensemble of Decision Trees is the average over all predictions in a regression setting, or a majority vote in a classification setting. A Random Forest additionally *decorrelates* the Decision Trees. To this end, during training of each Decision Tree and during each split, the algorithm is only allowed to consider a subset $m < p$ of prediction features as a potential split feature. A typical setting for the hyperparameter m is $m \approx \sqrt{p}$ [84]. The idea behind this restriction is the following: Suppose there is one very strong prediction feature. Without this restriction, it is likely that each individual Decision Tree will rely on this prediction feature as its top split. Consequently, most Decision Trees in the ensemble will be similar to each other and the predictions of the individual trees will be highly correlated [84]. Thus, the variance of the Random Forest compared to the variance of a single Decision Tree will not be substantially reduced and the overhead of training multiple Decision Trees will not lead to a substantial performance improvement.

Figure 2.12 shows a schematic overview for the training procedure of a Random Forest using the *Iris* dataset.

The number of Decision Trees to use in a Random Forest is highly dataset-dependent and should be individually optimized. Other hyperparameters for Random Forests include the hyperparameters of the individual Decision Trees and the number of features m that are used for each split. Typically, a global set of hyperparameters is determined for all Decision Trees in a Random Forest. The training and inference of a Random Forest can be easily parallelized, since all Decision Trees can be trained and queried independently.

In this thesis, we rely on the Random Forest regression model as implemented in the Python package *scikit-learn* [135].

Gradient Boosted Trees

Gradient Boosted Trees (*GBTs*) [59] is an example of a learning algorithm based on boosting. The core principle of GBTs is that each Decision Tree tries to predict the *pseudo-residuals* of the previous tree. This means that each tree tries to learn how to improve the overall prediction by predicting the accumulated error of its predecessors. The training consists of the following three steps: an initialization, B subsequent *boosting rounds* (each resulting in a single Decision Tree), and finally, the accumulation of all predicted adjustments of all boosting rounds to obtain the final model.

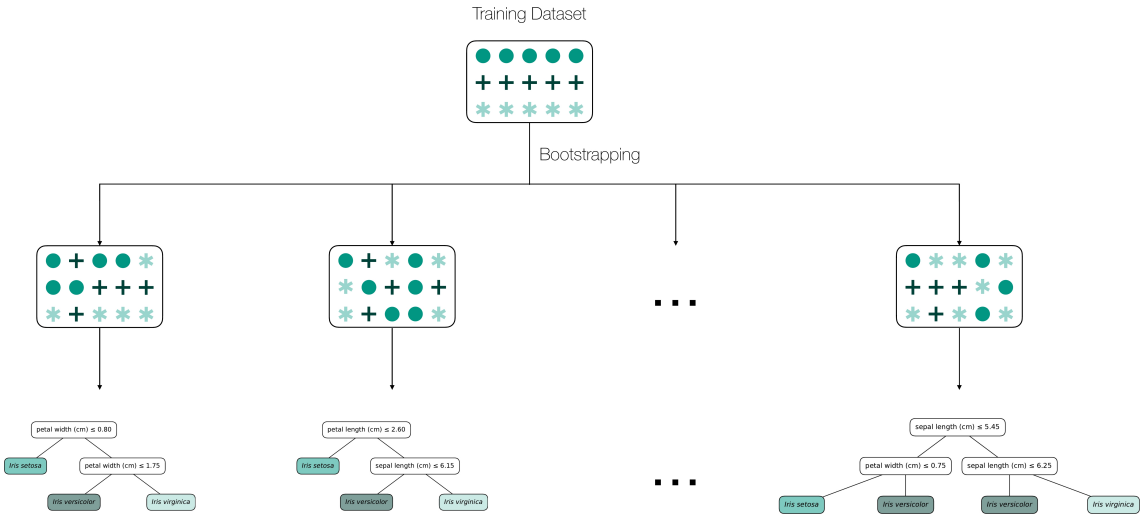


Figure 2.12: Schematic visualization of the training procedure of a Random Forest classifier for the *Iris* dataset. The independent Decision Trees are trained on bootstrapped datasets.

The training procedure is initialized using a constant *baseline* prediction for all samples. This baseline prediction is computed by minimizing the loss function

$$f^0(\boldsymbol{X}) = \arg\min_{\gamma} L(\boldsymbol{y}, \gamma). \tag{2.55}$$

For example, for regression tasks trained with respect to the MSE loss, the baseline prediction is simply the average of the target vector $\bar{y}$. For binary classification tasks trained using the BCE loss, the baseline prediction is the majority class.

This initialization is followed by B *boosting rounds*. Each boosting round b yields a single Decision Tree and consists of the following three steps:

1. Compute the pseudo-residuals $\boldsymbol{r}^b$ as the negative gradients of the loss function with respect to the prediction of the current model f^{b-1}:

$$\boldsymbol{r}^b = -\frac{\partial L(\boldsymbol{y}, f^{b-1}(\boldsymbol{X}))}{\partial f^{b-1}(\boldsymbol{X})}. \tag{2.56}$$

   This essentially measures to which extent the prediction of the current model needs to be adjusted to reduce the overall prediction loss.

2. Fit a Decision Tree to predict these pseudo-residuals

$$\hat{\boldsymbol{r}}^b \approx \boldsymbol{r}^b. \tag{2.57}$$

3. Update the model by adding the scaled predicted residuals of this new Decision Tree to the current prediction:

$$f^b(\boldsymbol{X}) = f^{b-1}(\boldsymbol{X}) + \lambda \hat{\boldsymbol{r}}^b. \tag{2.58}$$

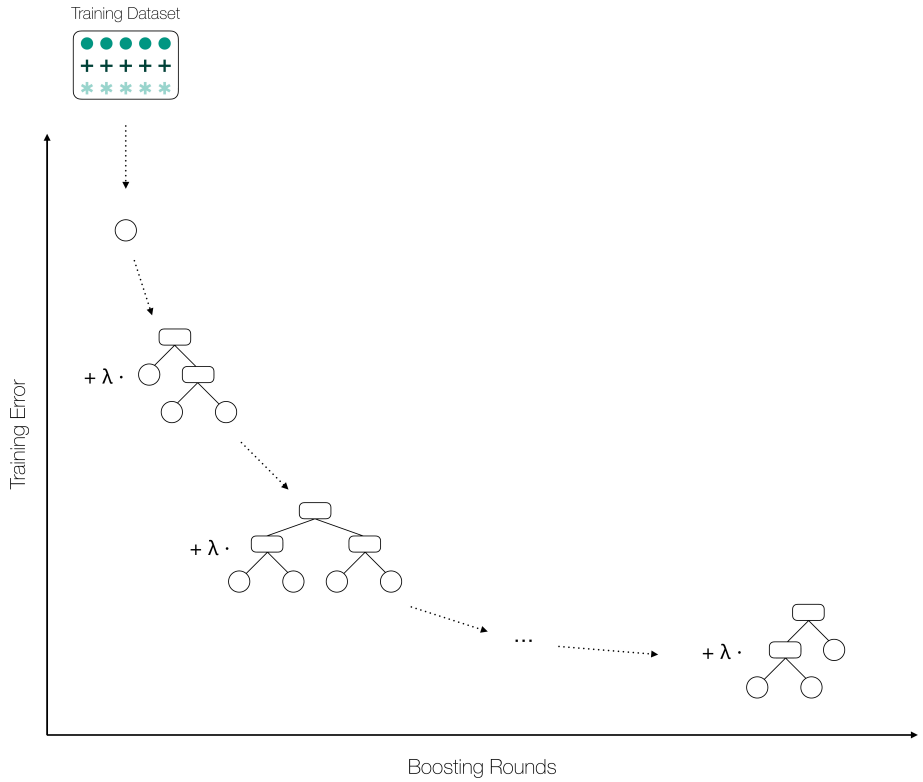


Figure 2.13: Schematic overview of the training process of a GBT model. The training is initialized with a constant prediction, and the subsequent boosting rounds each add a Decision Tree to refine the model's prediction and hence reduce the prediction error on the training data.

> The scale parameter $\lambda \leq 1$ determines the rate at which the overall GBT learns and is often called *learning rate*.

We obtain the final model f^B by accumulating the learned subsequent adjustments with respect to the baseline prediction:

$$f^B(\boldsymbol{X}) = f^0(\boldsymbol{X}) + \sum_{b=1}^{B} \lambda \hat{\boldsymbol{r}}^b. \tag{2.59}$$

Figure 2.13 shows a schematic overview of this training process for a GBT model.

While for regression, this algorithm can be implemented in a straightforward manner, classification requires additional steps for transforming predictions into probabilities. More precisely, in a classification setting, each individual Decision Tree is a regression tree that predicts log-odds and this prediction is only subsequently transformed into probabilities using the *sigmoid* function

$$s(z) = \frac{1}{1 + e^{-z}}, \tag{2.60}$$

and the final prediction is obtained by defining thresholds to determine the final class label.

The most important tuning parameters for GBTs are the number of boosting rounds and the learning rate λ [71]. Smaller λ values require more boosting rounds, resulting in an increased runtime for training. However, it was shown that learning rates $\lambda < 0.125$ lead to better generalization on unseen test data [59]. Similar to Random Forests, the hyperparameters for the individual Decision Trees are typically determined globally for all trees.

In contrast to Random Forests, the training of GBTs cannot be as easily parallelized due to the sequential training of Decision Trees. However, the prediction using a GBT can be easily parallelized since the Decision Trees obtained from the boosting rounds can be queried independently after training.

In this thesis, we rely on the implementation of GBTs for regression and classification as provided by the *LightGBM* Python package [89].

2.3.4.3 Convolutional Neural Network

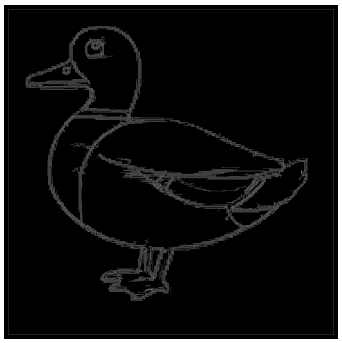
A *Convolutional Neural Network* (CNN) is a deep learning model that originated in computer vision and image processing [106]. Yet, since images are data matrices, CNNs are applicable to other domains as well by using other types of data, for instance, biological sequences.

A CNN comprises three main components: an input layer, a set of subsequent hidden layers, and an output layer. The input layer transforms the input data into the dimensionality expected by the first hidden layer. This typically is a *tensor* of size $H^0 \times W^0 \times C^0$, where H^0 and W^0 are the height and width of the matrix and C^0 is the number of *channels*. For example, with RGB images, the input is a $H^0 \times W^0 \times 3$ tensor that separates the three color channels.

The hidden layers are subsequent pairs of *convolution* and *pooling* layers. Each layer receives its predecessor's output of size $H^{i-1} \times W^{i-1} \times C^{i-1}$ as input. The idea of the subsequent convolution and pooling operations is to compute a condensed embedding of the information in the input tensor. The underlying operation of a convolutional layer is a *convolution* that operates on a two-dimensional matrix of the input tensor. A convolution *kernel* of size $k \times k$ with $k < H^{i-1}, W^{i-1}$ slides along the matrix with a stride s, and outputs the dot product of the matrix and the kernel at the current position. The resulting *feature map* highlights regions of the input that resemble the filter. For example, an edge detection kernel will highlight all edges in the resulting feature map. Figure 2.14 shows an example for an image of a mallard and the respective output of a convolution with a Canny edge detection filter [27]. A single convolution layer consists of a variety C^i of such kernels, and the resulting output tensor is of size $H^i \times W^i \times C^i$. The parameters of these kernels, that is, what exactly they detect in the input matrix, is learned during the training of the CNN.

Each convolution is followed by a non-linear *activation function*. Without such a non-linear activation function, a neural network can only learn linear relationships

(a) Input image of a mallard.



(b) Edges detected in the image after convolution using a Canny edge detection filter.

Figure 2.14: Example of an image convolution, applying a Canny edge detection filter to an image of a mallard.

in the data, since, regardless of the depth of the network, the prediction could be formulated as a linear combination of the input data. A non-linear activation function allows modeling more complex, non-linear relationships [84]. A typical choice is the *ReLU* (*rectified linear unit*) function $\mathrm{ReLU}(z) = max(0, z)$. A convolution layer is followed by a *pooling* layer. The pooling layer condenses the input matrix by summarizing blocks of information. For example, *max pooling* with a kernel size of k outputs the maximum value in a $k \times k$ block. Similar to the convolution kernel, this pooling kernel slides along the input matrix. Depending on the kernel size and the stride, the height and width of the input are substantially reduced, resulting in a compact representation of the information.

The final hidden layers are typically *fully connected* layers [106]. A fully connected layer receives a vector as input and outputs a vector. The input vector of the first fully connected layer is obtained by merely flattening the output of the last pooling layer. Each element in the output vector is computed as a weighted sum of all elements of the input vector. The weight for each connection between vector elements is learned during the training of the CNN.

For regression tasks, the output layer is a fully connected layer with a single value as output. This output is the final prediction of the CNN. For classification tasks, the output of the last fully connected layer is a K-dimensional vector with one dimension for each of the K target categories. This vector $\boldsymbol{z}$ needs to be transformed to a K-dimensional vector of class probabilities, which can be computed via the *softmax* activation function

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K. \tag{2.61}$$

The components of $\sigma(\boldsymbol{z})$ sum to 1, and $\sigma(\boldsymbol{z})_i$ can be interpreted as the predicted probability of the input datum being in category C_i.

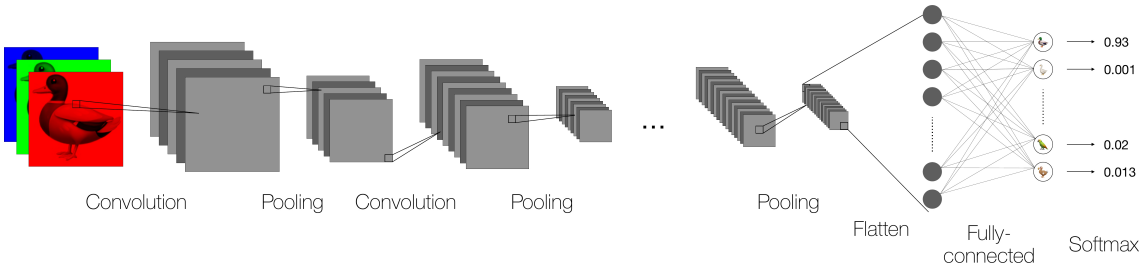


Figure 2.15: Schematic visualization of a CNN architecture for image classification.

Figure 2.15 shows a schematic overview of a CNN architecture for image classification. It comprises multiple hidden and subsequent convolution and pooling layers, one fully connected layer, and the final softmax activation function that categorizes the input image as *mallard* with 93% probability.

CNNs, and neural networks in general, are trained using *backpropagation* [106]. The backpropagation procedure propagates the prediction error backwards through the network and updates the parameters using the gradient of the error with respect to the current set of parameters.

Important hyperparameters to optimize in a CNN include the number of hidden layers, the kernel size, as well as the stride of the convolution and pooling kernels, and how many kernels to use in each layer.

Compared to the classical machine learning approaches discussed in the previous sections, a CNN requires only little pre-processing of the input data. This means that we do not need to manually engineer prediction features, as the CNN learns the relevant features by optimizing the kernel parameters and weights of the fully connected layer [106]. The drawback is that a CNN has substantially more parameters that need to be optimized during training. Consequently, CNNs require substantially more training data, and are more time- and resource-consuming during training, and inference. Yet, given enough training data, deep learning approaches tend to outperform classical machine learning methods [61].

2.3.5 Explainability

Understanding, interpreting, and trusting model predictions are important aspects of machine learning applications [107, 112]. Interpreting the importance of individual features is straightforward for linear and logistic regression, where the learned model parameters $\boldsymbol{\beta}$ directly quantify the contribution of each feature to the final model. With Decision Trees, the entire model can be visualized as a (binary) tree structure, making the decision process easy to understand. Interpreting ensemble methods such as Random Forests or GBTs requires more involved methods such as *feature importance* or *Shapley values* [157]. Explaining the decision process, let alone the learned features of deep learning models, is complicated due to the complex structure of these models and is beyond the scope of this thesis. In the following, we focus on feature importance and Shapley values for tree-based models. For a broad overview

of alternative approaches, as well as methods for explaining deep learning methods, we refer the interested reader to Linardatos *et al.* [107].

2.3.5.1 Feature Importance

Feature importance ranks the prediction features according to their contribution to the predictive power of the model. The more influence a feature has on the overall prediction, the higher its importance.

The *permutation importance* [23] measures the contribution of features based on a randomization approach. To determine a feature's impact on the model, the respective values in the training dataset are randomly shuffled. This random shuffling distorts the relationship between the feature and the target, and we can assess the impact on the model by measuring the degradation of model performance. Permutation importance is particularly useful when comparing multiple, individually trained machine learning models, since feature importance is evaluated based on the model's predictions, rather than the internal model structure. Hence, the permutation importance is a model-agnostic approach.

In contrast, the *gain-based importance* (also called *Gini importance*) measures feature importance using the internal structure of the underlying model and is only applicable to tree-based machine learning models. It measures the contribution of a feature to the model's prediction by quantifying the improvement under the used split criterion across all splits in all Decision Trees. The gain-based importance provides a straightforward interpretation of feature contributions to the overall model performance. This is particularly useful for understanding the decision-making process of a model.

2.3.5.2 Shapley Values

Feature importance provides a *global* explanation of feature contributions across all training samples. If, instead, we are interested in the contributions of specific feature values for a single training datum, feature importances provide little insight. However, Shapley values have been specifically designed to answer this question.

Shapley values originate from cooperative game theory [157]. The goal is to distribute the payout of a game fairly among cooperative players based on their contribution. Translated to machine learning, the game is the prediction of a single datum, the players are the feature values $x_i^1, \ldots, x_i^p$ of the training sample $\boldsymbol{x}_i$, and the payout is the difference in prediction $\hat{y}_i$ for $\boldsymbol{x}_i$ compared to the average prediction across all training samples. A naïve approach to measuring the contribution of feature j with feature value x_i^j would be to replace x_i^j with randomly sampled values from the training dataset and average the resulting differences in payouts. However, this approach is likely to generate unrealistic samples, as the resulting combination of feature values may be improbable. Additionally, this approach assumes features to be independent of each other, which is rarely the case in real-world applications.

Instead, Shapley values rely on the concept of *marginal contributions*. The marginal contribution is the change in the prediction when a particular feature is added to

a subset of other features. To calculate the Shapley value for a certain feature, we first consider all possible subsets of the remaining features and determine the marginal contribution of the target feature by observing the difference in the model's prediction with and without that feature in each subset. The Shapley value for a feature is then calculated as the weighted average of these marginal contributions over all possible subsets. Shapley values satisfy four important aspects of payout attribution: Efficiency, Symmetry, Dummy, and Additivity. *Efficiency* means that the Shapley values of all features p for a datum $\boldsymbol{x}_i$ sum to the difference in prediction $\hat{y}_i$ and the average prediction, meaning the payout is fully divided among players. *Symmetry* ensures that the Shapley values of two features are identical if, and only if, their contribution is identical. Shapley values further ensure that a *dummy* feature which does not influence the prediction has a Shapley value of 0. And finally, Shapley values are *additive*, meaning, for example, that the Shapley value of a feature in a Random Forest can be computed as the average Shapley Value over all Decision trees.

Shapley values are inherently designed to explain the prediction of a single datum, and a Shapley value is only applicable to a certain training datum with its distinct combination of feature values. We can obtain a feature-importance-like measure for a set of samples by averaging the per-feature Shapley values across all samples.

With p prediction features, Shapley values need to consider all possible 2^p combinations, making Shapley values computationally very costly, especially for large numbers of prediction features. Shapley values are, in general, model agnostic, but leveraging model structure can help to reduce the computational cost [112]. In this thesis, we rely on Shapley values as implemented in the Python package *SHAP* [112]. We specifically rely on the *TreeExplainer* method for efficient computation of Shapley values for tree-based models [113].

3. Pythia: Predicting the Difficulty of Phylogenetic Analyses

This chapter is derived from the open-access publications:

> **Julia Haag**, Dimitri Höhler, Ben Bettisworth and Alexandros Stamatakis. "From Easy to Hopeless - Predicting the Difficulty of Phylogenetic Analyses." *Molecular Biology And Evolution*, Volume 39, Issue 12, December 2022. https://doi.org/10.1093/molbev/msac254
>
> **Julia Haag** and Alexandros Stamatakis. "Pythia 2.0: New Data, New Prediction Model, New Features." *bioRxiv*, 2025. https://doi.org/10.1101/2025.03.25.645182

Julia Haag designed, implemented, and evaluated all presented methods. All text and figures in this chapter were created by Julia Haag. Dimitri Höhler, Ben Bettisworth, and Alexandros Stamatakis provided background knowledge, discussion, and feedback.

Note that the first publication is peer-reviewed. The second publication is a preprint, which we do not plan to submit for peer review. It primarily offers an overview of all changes and improvements made to Pythia since our initial publication to keep users informed on the latest updates.

3.1 Background and Motivation

The goal of a phylogenetic inference is to find the phylogenetic tree that best explains the given biological sequence data. Since the number of possible tree topologies grows super-exponentially with the number of taxa, one cannot compute and

score every possible tree topology. Instead, one deploys tree inference heuristics that explore the tree space, hoping to find a tree with a 'good' score, for example, under the Maximum Likelihood (ML) criterion [195]. However, these heuristics do not guarantee that the tree inference will converge to the *globally* optimal tree. Therefore, under ML, one typically infers multiple trees and subsequently summarizes the inferred, *locally* optimal trees via a consensus tree. One can observe that for some datasets, all individual, independent ML tree searches converge to topologically highly similar trees. This suggests that the likelihood surface of such datasets exhibits a single likelihood peak, yielding the dataset easy to analyze. For other datasets, one observes that the independent tree inferences converge to multiple topologically highly distinct, yet, regarding their ML score, statistically indistinguishable, locally optimal trees. These datasets are hence difficult to analyze, and we say that they exhibit a rugged likelihood surface. This diverse behavior of phylogenetic tree searches has already been reported in several publications (Lakner *et al.* [100]; Stamatakis [166]; Morel *et al.* [125]). In general, the more tree inferences we perform, the better our understanding of the dataset's behavior and coverage of the respective tree space will become. However, under ML, inferring a single tree can already require multiple hours or even days of CPU time. To save time and resources, an optimal analysis setup will perform as few tree inferences as necessary. For easy-to-analyze datasets with a single likelihood peak, we require fewer and less involved tree search heuristics and bootstrap replicate searches to sample the tree space adequately, in contrast to difficult-to-analyze datasets with rugged likelihood surfaces. To the best of our knowledge, and despite anecdotal reports on the behavior of difficult datasets, there does not yet exist a *quantifiable* definition of dataset difficulty that captures the behavior of ML tree searches on a given input dataset.

In order to speedup ML tree inferences, researchers have developed elaborate ML tree inference tools that combine multiple search strategies to reduce the risk of becoming stuck in local optima. There also exist early stopping criteria to determine whether the tree inference has converged. Such early stopping methods either deploy ad hoc or statistical criteria to terminate the tree inference. For example, the ML tree inference software FastTree [142] relies on a maximum number of topology optimization iterations as a function of the number of sequences in the dataset. The ML software RAxML [167] implements an early stopping criterion based on the topological distance between the respective best trees found in two consecutive optimization cycles [166]. Vinh and von Haeseler [181] propose an estimation criterion that determines with 95% confidence whether continuing the tree inference will yield a better tree than the currently best tree. However, early stopping criteria only determine the convergence of the current tree search, but they do evidently not guarantee that the search has converged to the globally optimal tree. Thus, to better characterize and explore the tree search space, additional tree inferences and subsequent a posteriori analyses are required. In contrast, assessing the expected behavior of a dataset *before* conducting compute-intensive tree inferences allows for a more informed decision on the most appropriate tree inference and post-analysis strategy. It also allows users to reassemble/modify difficult datasets, as these will most likely require resource-intensive analyses that yield contradicting, yet almost

equally likely, tree topologies with low confidence. Several methods have already been developed to assess the information content of datasets *prior* to tree inference, the most prominent example being the treelikeness of a dataset [12, 115, 187]. A simple and fast-to-compute metric is the sites-over-taxa ratio. For instance, [147] conclude that a higher phylogenetic inference accuracy can be achieved by increasing the MSA length, rather than including more taxa/sequences. A more involved method was proposed by [81]. The authors suggest the use of δ-plots, that is histograms, based on all quartet distances induced by the Multiple Sequence Alignment (MSA). However, computing the δ-plots is time-intensive due to the computational complexity of $\mathcal{O}(N^4)$, where N is the number of taxa in the MSA. Misof *et al.* [123] provide an overview of various methods for calculating the treelikeness, before a phylogenetic analysis. The authors acknowledge that the considered treelikeness estimation methods capture certain aspects of the MSAs. However, they conclude that none of them sufficiently informs the user about the expected behavior of phylogenetic analyses in general, and suggest further research in this area.

With our work, we initially introduce a quantification of the degree of difficulty based on the result of 100 ML tree inferences per MSA. We demonstrate that this quantification adequately represents the behavior of the ML searches on the dataset. Since executing 100 ML tree searches is computationally prohibitive in general, we train a machine learning regression model that can predict the difficulty of a given MSA that is exclusively based on MSA attributes and fast and thus substantially less expensive tree inferences under maximum parsimony (MP) [44, 52]. By extracting multiple simple and fast-to-compute attributes, such as the sites-over-taxa ratio, and by deploying machine learning, we devise an accurate difficulty predictor called *Pythia*. In our initial work, `Pythia 0.0`, that was published in Haag *et al.* [66], we used Random Forest regression trained on 3250 MSAs, resulting in a high accuracy with an MAE of 0.09 and an MAPE of 2.9 %. With our latest, updated `Pythia 2.0` version, presented in Haag and Stamatakis [65], we employed Gradient Boosted Tree regression trained on 10 461 MSAs, resulting in a slight prediction improvement with an MAE of 0.08 and an MAPE of 2.2%.

Our latest `Pythia 2.0` version is on average approximately 2 times faster than `Pythia 0.0` and 20 times faster than a *single* ML tree inference. Pythia predicts the difficulty of a dataset on a scale ranging between 0.0 (easy) to 1.0 (difficult).

In contrast to the aforementioned early stopping criteria that can be applied during ML searches, Pythia informs the user about the expected behavior of the MSA under ML phylogenetic analysis *prior* to conducting any ML phylogenetic inference. Thereby, users can make informed decisions on the most appropriate ML analysis and post-analysis strategy. This includes, for example, a careful consideration of the number of required independent, resource-intensive, tree searches as a function of the difficulty. Furthermore, for difficult MSAs, the user will be able to improve the informativeness of the MSA, for example, by increasing sequence length or removing sequences, to assemble an MSA that is easier to analyze. Thereby, one can save valuable time and resources by not performing tree inferences on difficult MSAs. We therefore suggest that a difficulty analysis with Pythia should be conducted at the

beginning of any ML phylogenetic analysis. Note that the predicted difficulty does not directly predict the number of tree inferences required to sufficiently sample the tree space, as this number also depends on the implemented tree inference heuristic.

Pythia is available as open-source software libraries in C and Python. Both libraries include the trained prediction model and the computation of the required prediction features. The C library CPythia is an addition to the COre RAXml LIBrary (Coraxlib) [43] and is available at `https://github.com/tschuelia/CPythia`. Additionally, we provide PyPythia, a lightweight, stand-alone Python library, including a respective command line interface. PyPythia is available at `https://github.com/tschuelia/PyPythia`, including a thorough documentation of the command line interface and the Python library.

3.2 Methods

We formulate the difficulty prediction challenge as a supervised regression task. The goal is to predict the difficulty on a scale ranging between 0.0 (easy) to 1.0 (difficult). We face two main challenges: (i) obtaining ground-truth difficulties that represent the actual difficulty of the training data and (ii) obtaining a sufficiently large set of MSAs to train Pythia on, ideally comprising empirical MSAs.

In the following, we motivate and present a *difficulty* quantification that captures the "ruggedness" of the likelihood surface on a scale from 0 to 1. We further present our setup to train a regression model that accurately predicts this difficulty. We focus on the prediction features, the training datasets, and our machine learning models for both, our first prediction model `Pythia 0.0` and our latest model `Pythia 2.0`. `Pythia 0.0` corresponds to the initial version of Pythia published in Haag *et al.* [66]. `Pythia 2.0` corresponds to our latest model published in Haag and Stamatakis [65]. It includes all changes and improvements we introduced since our initial publication in 2022.

3.2.1 Difficulty Quantification

To train a difficulty predictor, we require a reliable ground-truth label for each training datum. To obtain such labels, we must initially quantify the difficulty. To stringently quantify the difficulty of an MSA, we would have to explore the entire tree space. Since this is computationally not feasible, we need to rely on a heuristic definition. Our heuristic to quantify the difficulty is based on 100 ML tree inferences we performed using RAxML-NG [96]. First, we infer $N_{\text{all}} := 100$ ML trees and compute the average pairwise RF-Distance (see Section 2.1.4.1) between all trees (RF_{all}), as well as the number of unique topologies among the 100 inferred trees (N^*_{all}). Then we determine the best tree among the 100 inferred trees according to the log-likelihood, and compare all trees to this best tree using statistical significance tests (see Section 2.1.4.2). We assign trees that are not significantly worse than the best tree to a so-called *plausible tree set*. In our analyses, we use the statistical significance tests as implemented in the IQ-TREE software package [121]. Due to the continuing debate about the most appropriate significance test for comparing

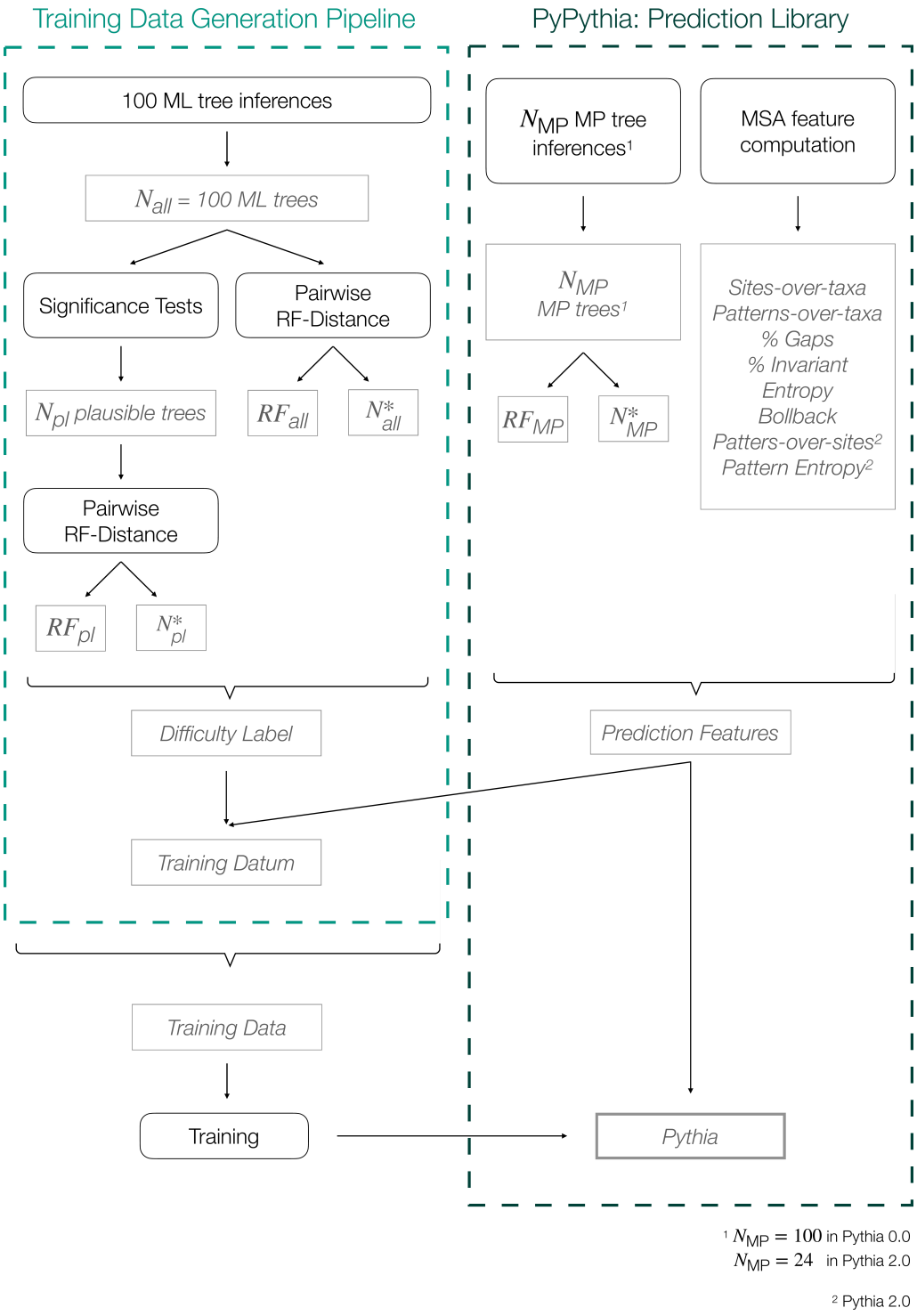


Figure 3.1: Schematic depiction of the training data generation procedure. For each MSA, we compute the difficulty label based on our difficulty quantification using our training data generation pipeline (left dashed box). We further compute the prediction features using our Python prediction library PyPythia (right dashed box). Using the difficulty label and the corresponding prediction features for all MSAs in our training data, we train Pythia.

phylogenetic trees, we use the approach suggested by Morel *et al.* [125] and only include trees that pass *all* significance tests in the plausible tree set. We further refer to the number of trees in this plausible tree set as N_{pl}. We compute the average pairwise relative RF-Distances between trees in the plausible tree set (RF_{pl}), as well as the number of unique topologies (N^*_{pl}). Finally, we compute the difficulty of the dataset based on the following formula:

$$\text{difficulty} = \frac{1}{5} \cdot \Bigg[RF_{\text{all}} + RF_{\text{pl}} \tag{3.1}$$

$$+ \frac{N^*_{\text{all}}}{N_{\text{all}}} + \frac{N^*_{\text{pl}}}{N_{\text{pl}}} \tag{3.2}$$

$$+ \left(1 - \frac{N_{\text{pl}}}{N_{\text{all}}}\right) \Bigg] \tag{3.3}$$

The rationale for expression (3.1) is that, if the RF-Distance is high, the tree space comprises multiple distinct, locally optimal tree topologies which characterize a dataset that is difficult to analyze. With expression (3.2) the rationale is that the tree surface becomes more rugged, the more distinct locally optimal tree topologies the tree inference yields, and the more tree topologies are not significantly different from the best tree. Finally, the rationale for expression (3.3) is that, the more tree inferences yield a plausible tree, the more informative the MSA will be about the underlying evolutionary process and the easier this MSA will be to analyze. Each term is a value between 0.0 and 1.0, leading to an average value between 0.0 and 1.0 that quantifies the overall difficulty.

For each MSA in our training data, we compute the difficulty according to this definition. To this end, we implemented a training data generation pipeline that automatically performs all required tree inferences, statistical tests, and computes the difficulty label alongside the features required for training Pythia. We implement this pipeline using the Snakemake workflow management system [94] and Python 3. The pipeline code is available at `https://github.com/tschuelia/difficulty-prediction-training-data`.

Due to the lack of absolute ground-truth labels, we need to rely on the inferred difficulty labels for training Pythia. In Section 3.3.1 below, we provide a thorough analysis to demonstrate that our difficulty quantification can accurately capture the tree search complexity under ML-based phylogenetic inference, and thus provides a reasonable training target for machine-learning models.

3.2.2 Model Training

Figure 3.1 depicts the workflow for generating the training data for our Pythia difficulty predictor. For each MSA, we compute the difficulty according to the above definition as ground-truth label for supervised training using the training data generation pipeline. We compute the corresponding prediction features using

Feature	Explanation
MSA Attributes	
Sites-over-taxa ratio	The number of sites divided by the number of taxa: M/N.
Patterns-over-taxa ratio	The number of unique site patterns divided by the number of taxa: P/N.
% Invariant	The proportion of fully conserved sites in the MSA.
% Gaps	The proportion of gaps in the MSA.
MSA Information Content	
Entropy	The Shannon Entropy [156] H(MSA) to capture the information content in the MSA. See below for further details on how we compute H(MSA).
Bollback Multinomial	Multinomial test statistic T(MSA) based on the frequency of site patterns [20]. See below for further details on how we compute T(MSA).
Treelikeness	The treelikeness score [81] is a measure of phylogenetic signal in the MSA. See below for further details on how we compute this score.
ML-based Features	
# SPR rounds	RAxML-NG [96] optimizes the tree topology of an initial starting tree using Subtree Pruning and Regrafting (SPR) moves (see Section 2.1.3.3). One iteration of pruning and regrafting all possible subtrees in the current tree is called an *SPR round*. This feature counts the number of SPR rounds RAxML-NG performs during a single ML tree inference.
RF_{ML}	RF-Distance between the starting tree topology and the inferred ML tree topology.
$\text{brlen}_{\text{min/max/avg/std/sum}}$	Minimum, maximum, average, standard deviation, and sum over all branch lengths in the inferred ML tree.
MP-based Features	
RF_{MP}	Average pairwise relative RF-Distance between the inferred MP trees.
N^*_{MP}	Proportion of unique tree topologies among the inferred MP trees.

Table 3.1: Overview and explanation of prediction features we considered for training `Pythia 0.0`. N denotes the number of taxa in the MSA, M the number of sites, and P the number of unique site patterns.

our Python library *PyPythia*. The set of prediction features and the corresponding difficulty label form our training data. We split the training data into two sets: a training dataset used for training the model and a test set that is exclusively used for evaluating the predictive power of the difficulty predictor. To ensure an even distribution of difficulty labels in the training and test sets, we deploy stratified

sampling. Stratified sampling splits all difficulty labels into disjoint subsets and draws random samples from each subset independently. Note that we deployed different training and splitting procedures in `Pythia 0.0` and `Pythia 2.0` which we will explain in more detail below.

In the following, we provide further details on the prediction features and the collections of MSAs we use for training Pythia. Additionally, we explain the training setup of our initial model `Pythia 0.0`, as well as our latest model `Pythia 2.0`.

3.2.2.1 Prediction Features

In our preliminary experiments for `Pythia 0.0`, we assessed a plethora of distinct possible prediction features. In the following, we present all the features we implemented and analyzed, and motivate the selection of the eight features that we finally used to train `Pythia 0.0`. We further explain the prediction feature changes and additions introduced in `Pythia 2.0`.

For `Pythia 0.0`, we considered four feature categories. Table 3.1 provides an overview and explanation of all features. The first category comprises MSA attributes that are fast to compute, such as the proportion of gaps in the MSA. The second category captures information content. Together, both categories aim to capture the phylogenetic signal in the MSA. We further analyzed features based on a single ML tree inference. While with Pythia we aim to reduce the time- and resource-consumption of (ML) tree inferences, the hypothesis was that using features based on a *single* ML tree can yield useful information about *multiple* ML tree inferences. Finally, we include two features that are based on trees inferred under the fast-to-compute MP criterion. In `Pythia 0.0`, we relied on 100 MP trees. Following an extensive analysis that we describe further below, we reduced the number of MP trees in `Pythia 2.0` to 24.

The Shannon Entropy [156] measures the information value of data. We compute the Entropy for an MSA with N taxa as average Shannon Entropy over all M MSA sites:

$$H(\text{MSA}) = \frac{1}{M} \sum_{j=1}^{M} H(\text{site}_j) \tag{3.4}$$

$$\text{with} \quad H(\text{site}_j) = -\sum_{i=1}^{N} p(s_i^j) \cdot log(p(s_i^j)), \tag{3.5}$$

where $p(s_i^j)$ denotes the relative frequency of the respective character at the j-th MSA site.

Bollback [20] designed a multinomial test statistic to quantify the frequency of site patterns. We compute the Bollback Multinomial test statistic as:

$$T(\text{MSA}) = \left(\sum_{i=1}^{P} P_{\xi(i)} \cdot ln(P_{\xi(i)}) \right) - M \cdot ln(M), \tag{3.6}$$

where $\xi(i)$ denotes the i-th unique pattern and $P_{\xi(i)}$ the number of times this pattern occurs.

The treelikeness score [81] is designed to quantify the phylogenetic signal in the data. The treelikeness score is based on a matrix of pairwise distances between all sequences in the MSA. Let N be the number of taxa, and $\mathbf{\Delta} \in N \times N$ the pair-wise distance matrix with entries $\delta_{i,j}$; $0 \leq i, j \leq N$. For a set of four taxa (a *quartet*) $q = (x, y, u, v)$, we compute the treelikeness as:

$$\delta_q = \frac{\delta_{xv|yu} - \delta_{xu|yv}}{\delta_{xv|yu} - \delta_{xy|uv}} \tag{3.7}$$

$$\text{with} \quad \delta_{xy|uv} = \delta_{xy} + \delta_{uv} \tag{3.8}$$

$$\text{and} \quad \delta_{xy|uv} \leq \delta_{xu|yv} \leq \delta_{xv|yu} \tag{3.9}$$

The lower the score δ_q, the stronger is the phylogenetic signal in the quartet q. For a set of N taxa, we compute this δ_q-score for every possible quartet q. The treelikeness of the entire data is then computed as the mean over all δ_q-scores. Due to the computational complexity on MSAs with many taxa ($N > 100$), we follow the suggestion by Holland *et al.* [81] and compute the δ_q-scores based on a random sample of 100 taxa.

To minimize the runtime of Pythia's difficulty prediction, we analyzed the runtime of computing each feature for all MSAs in our training data, as well as the feature importance for the prediction. Note that we conducted the following analyses using the training dataset of `Pythia 0.0` comprising 3250 MSAs.

To determine the optimal set of prediction features, we trained a Random Forest regression model using all presented features and considered the respective feature importances, as well as the runtimes to obtain the respective features. Table 3.2 shows the permutation importance and the runtime relative to a single ML tree inference using RAxML-NG for each of the presented features. For benchmarking the runtimes of the feature computation, we used the implementation in our Python library *PyPythia* (version 0.0). Note that the ML-based features require a complete ML tree search. Consequently, the relative computing time for these features is $\geq 100\%$. For obtaining the number of taxa, sites, patterns, the proportion of gaps and invariant sites, we used the RAxML-NG `--parse` option. Since RAxML-NG log files contain but a few lines, parsing the respective files using Python is fast (runtime $\ll 10\,\text{ms}$). We therefore omitted this additional runtime contribution in our assessment.

Based on these analyses, we selected the following subset of eight features, providing a good trade-off between accuracy and runtime: sites-over-taxa ratio, patterns-over-taxa ratio, % invariant, % gaps, Entropy, Bollback Multinomial, RF_{MP}, and N^*_{MP}. In particular, omitting the ML-based features and the treelikeness score improves the runtime substantially. Thus, despite the substantial feature importances of the branch-length statistics (especially the sum over all branch lengths), we decided against using them for training Pythia. To quantify the impact of this choice on

Feature	Importance in percent	Relative runtime $\mu \pm \sigma$ (median) in percent
MSA Attributes		
Sites-over-taxa ratio	3.5%	$1.1 \pm 3.1\%$ (0.2%)
Patterns-over-taxa ratio	3.0%	
% Invariant	1.0%	
% Gaps	1.6%	
MSA Information Content		
Entropy	5.2%	$8.8 \pm 65.8\%$ (1.5%)
Bollback Multinomial	3.1%	$3.3 \pm 16.6\%$(1.5%)
Treelikeness	3.8%	$450 \pm 965\%$(254.3%)
ML-based Features		
# SPR rounds	5.7%	$100.0 \pm 0.0\%$ (100.0%)
RF_{ML}	3.5%	$100.8 \pm 2.5\%$ (100.1%)
$\text{brlen}_{\text{min}}$	0.2%	$100.1 \pm 0.2\%$ (100.0%)
$\text{brlen}_{\text{max}}$	0.9%	
$\text{brlen}_{\text{avg}}$	2.5%	
$\text{brlen}_{\text{std}}$	1.9%	
$\text{brlen}_{\text{sum}}$	9.3%	
MP-based Features		
RF_{MP}	54.1%	$8.2 \pm 13.4\%$ (3.8%)
N^*_{MP}	0.7%	

Table 3.2: Permutation importance and runtime relative to a single ML tree inference in RAxML-NG for all features we considered for `Pythia 0.0`.

the overall prediction accuracy, we compared a Random Forest regressor trained using *all* features to a Random Forest regressor only trained on the subset of eight features. The model relying on all features showed an MAE of 0.08 (MAPE = 2.1%). The model relying on the subset of eight features showed only a slight decrease in performance, with an MAE of 0.09 (MAPE = 2.9%).

New Features in Pythia 2.0

In `Pythia 2.0`, we included two additional features: the *patterns-over-sites* ratio as an additional MSA attribute and the *Pattern Entropy* as an additional approach to capturing the information content of the MSA. The patterns-over-sites ratio is simply the number of unique sites patterns in the MSA divided by the total number of sites. The Pattern Entropy is an entropy-like measurement based on the number and frequency of unique site patterns in the MSA. We compute the Pattern Entropy H_P as

$$H_P(\text{MSA}) = \sum_{i=1}^{n} P_{\xi(i)} \cdot ln(P_{\xi(i)}),$$

where $\xi(i)$ denotes the i-th unique pattern and $P_{\xi(i)}$ the number of times this pattern occurs. Note that the computation of this feature is closely related to the *Bollback*

Multinomial feature. However, we expect it to be less biased by the total number of sites in the MSA.

Reducing the Number of Maximum Parsimony Trees in Pythia 2.0

To improve upon the runtime of Pythia's feature computations, for `Pythia 2.0`, we aimed to reduce the number of MP trees required for computing the RF_{MP} and N^*_{MP} features. In `Pythia 2.0`, we only rely on 24 inferred trees, compared to the 100 trees in `Pythia 0.0`. In the following, we detail the analyses leading to the conclusion that 24 constitutes the optimal number of MP tree inferences.

The MP criterion is $\mathcal{NP}$-hard [55]. Consequently, implementing tree inferences under this criterion requires heuristic algorithms. For instance, RAxML-NG implements a randomized stepwise addition order algorithm to infer MP trees. A consequence of these heuristics is that different initializations may result in topologically distinct inferred trees, and can consequently yield variations in the average relative pairwise RF-Distance between a set of inferred trees (RF_{MP}). In `Pythia 0.0`, we inferred 100 MP trees, simply because we based the ground-truth difficulty on 100 ML trees. However, we observed that the runtime of `Pythia 0.0` is dominated by the MP tree inference runtime (approximately 50% of total runtime). Thus, to reduce the computational overhead of the MP-based features in `Pythia 2.0`, we aimed to reduce the number of inferred MP trees.

Inferring 100 MP trees under varying seeds used for the random number generation will likely result in a variation of the resulting RF_{MP} feature value. We can leverage this expected variation as a baseline to find a lower number of MP trees m for the MP-based feature computation. To determine this baseline variation, we inferred 20 sets of 100 MP trees each, using a different random seed i for each set. We then computed the average pairwise RF-Distance RF^i_{100} within each set of 100 trees inferred under seed i. Finally, we computed the average standard deviation σ_{100} of absolute pairwise differences between the RF-Distances of different seeds:

$$\sigma_{100} = \sqrt{\frac{1}{190} \sum_{i,j,i \neq j} var\left(|RF^i_{100} - RF^j_{100}|\right)}. \tag{3.10}$$

Note that the mean of a standard deviation is computed as the square root of the average variance within all $\frac{20 \cdot 19}{2} = 190$ unique pairs of RF-Distance values inferred under 20 distinct seeds.

Using this baseline, our goal is to find the minimum number of MP trees m such that the average standard deviation of differences in RF-Distance when using only m trees compared to using 100 trees ($\sigma_{m,100}$) is less than σ_{100}. To this end, we computed the RF-Distance between subsets of $m = 3 \ldots 99$ trees sampled from the 100 trees inferred under seed i for all 20 seeds used for the baseline computation. Then, for each m, we compute the average standard deviation $\sigma_{m,100}$ of absolute pairwise differences between the RF-Distance using m MP trees versus using 100 MP trees for all seeds i:

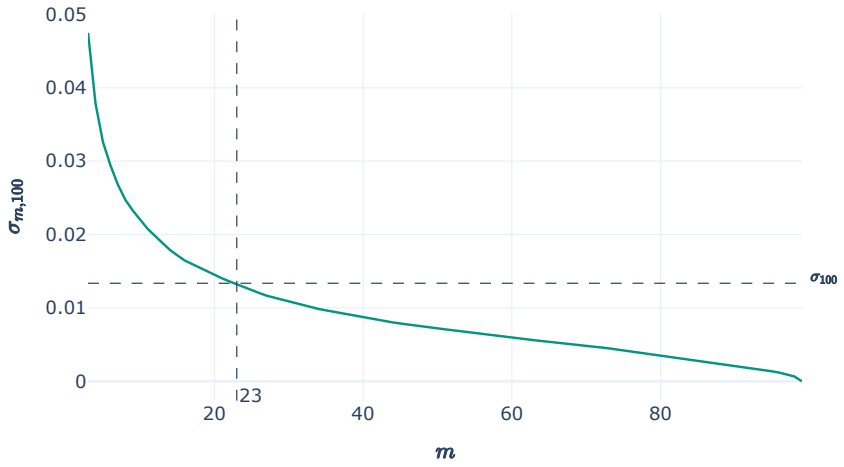


Figure 3.2: Visualization of $\sigma_{m,100}$ and σ_m for $m = 3 \ldots 99$ averaged across 1000 MSAs. The dashed horizontal line indicates σ_{100} and the dashed vertical line the intersection of $\sigma_{m,100}$ with σ_{100} at 23.

$$\sigma_{m,100} = \sqrt{\frac{1}{20} \sum_i var\left(|RF_m^i - RF_{100}^i|\right)}. \tag{3.11}$$

Finally, we visualize $\sigma_{m,100}$ as a function of the number of trees m in Figure 3.2. The x-axis shows the number of trees ($m = 3 \ldots 99$), and the y-axis shows the average standard deviation $\sigma_{m,100}$. The dashed horizontal line highlights σ_{100}, corresponding to the baseline variance between seeds when inferring 100 MP trees. Note that we averaged both values across all 1000 MSAs for this visualization. The intersection of $\sigma_{m,100}$ with σ_{100} corresponds to the minimum number of MP trees that are required such that the difference in RF_{MP} using only m trees compared to 100 trees is less than or equal to the variance of differences in RF_{MP} when using 100 trees but varying the random seed. As indicated by the dashed vertical line, $\sigma_{23,100} < \sigma_{100}$. This means that inferring 23 MP trees constitutes the optimal tradeoff between approximation accuracy and runtime. We decided to use one additional tree and infer 24 MP trees instead, as 24 inferences are easily parallelizable in machines with varying core counts (divisible by 2, 3, 4, 6, 8, and 12).

3.2.2.2 Training Data

We trained `Pythia 0.0` on 3250 empirical MSAs obtained from TreeBASE [138] comprising 74% DNA MSAs and 26% amino acid (AA) MSAs. For our latest predictor, `Pythia 2.0`, we increased the training dataset to 10 461 MSAs, using MSAs from TreeBASE and the RAxML Grove database [79]. Note that the publicly available RAxML Grove database only contains phylogenetic trees. The underlying, fully

anonymized MSAs are only available internally within our research group, and we only use these MSAs to compute the features required for training Pythia. This new training dataset comprises 87% DNA, 9% AA, and 4% morphological MSAs. Since we explicitly include morphological MSAs, `Pythia 2.0` can now accurately predict the difficulty for this data type as well. Note that we provide a single predictor for all data types, as we observe no substantial prediction accuracy differences when using a combined predictor compared to using an independent, dedicated predictor for each data type separately (see Section 3.3.2 below).

In principle, using simulated data would allow us to increase the training data size. However, since simulating data that behaves analogously to empirical data under ML tree inferences constitutes a challenging task (see Höhler *et al.* [79], Trost *et al.* [178], and Chapter 4), we decided against using any simulated data for training Pythia.

Figure 3.3 visualizes the distribution of ground-truth difficulty labels and feature values in the training datasets for `Pythia 0.0` and `Pythia 2.0`. For better visualization, we removed values above the 95th percentile for the patterns-over-taxa, sites-over-taxa, Entropy, and Pattern Entropy features. We removed values below the 5th percentile for the Bollback Multinomial feature.

In the training data of `Pythia 0.0`, 46% of the MSAs are very easy, with a ground-truth difficulty of less than or equal to 0.1. Only 9% of MSAs are difficult (difficulty 0.7 or higher). With our training data for `Pythia 2.0`, we still observe an abundance of very easy MSAs (36%). We were nonetheless able to increase the number of difficult MSAs to 1401 (13%).

3.2.2.3 Prediction Models and Training

Pythia 0.0

We trained and evaluated `Pythia 0.0` using an 80/20 train-test split approach on the initial training dataset of 3250 MSAs. This means that we used 80% of the MSAs for training, and the remaining 20% for evaluation.

During our initial experiments, we trained distinct regression algorithms and compared their predictive power according to the MAE and the MAPE (see Section 2.3.3 for details on these metrics). We trained *linear regression*, *lasso regression* [174], *Adaptive Boosting* (AdaBoost) [57], *Support Vector regression* [21] and *Random Forest regression* [75] models. We trained each model using an 80/20 train-test-split approach, and the respective implementation in the *scikit-learn* Python package [135]. Additionally, we used dummy regressors as a baseline. These dummy regressors do not learn to distinguish the data, but predict a value based on predefined rules. We used three dummy regressors: one that always predicts the average difficulty of the training set, one that always predicts the 75th percentile of the training set, and one that predicts 0.5 for each dataset. Table 3.3 shows the MAE and the MAPE for all trained regression models. The Random Forest regressor outperforms all other models according to all metrics. Using the dummy regressors as a baseline, we observe that the trained regressors learned to predict dataset difficulty to some extent.

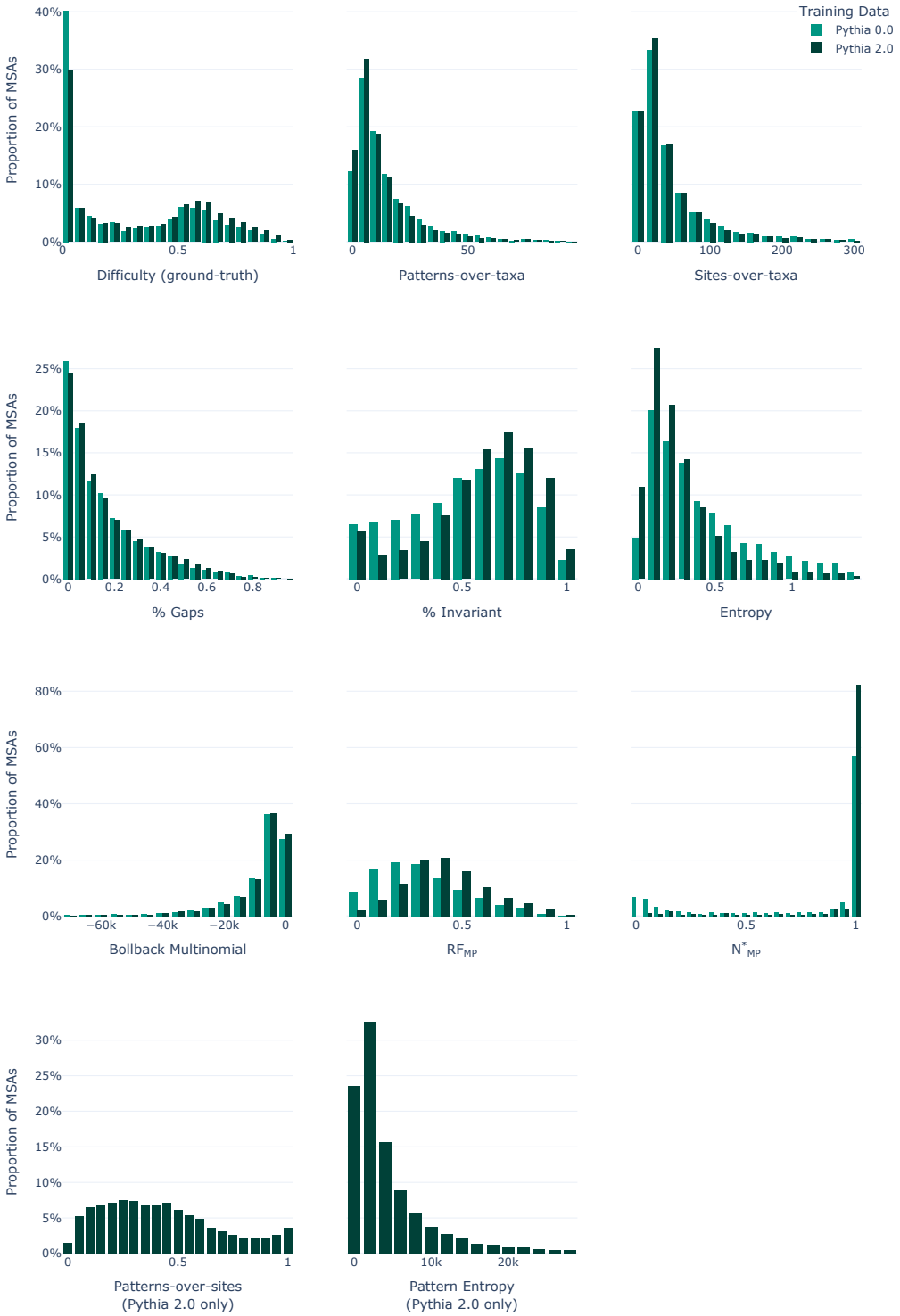


Figure 3.3: Distribution of ground-truth difficulty label and feature values in the training datasets for `Pythia 0.0` and `Pythia 2.0`. Note that we only introduced the patterns-over-sites and Pattern Entropy features in `Pythia 2.0`.

Algorithm	MAE	MAPE
Linear regression	0.14	7.2%
AdaBoost	0.15	9.7%
Lasso regression	0.26	18%
Support Vector regression	0.25	12.1%
Random Forest regressor	**0.09**	**2.9%**
Dummy regressor (average)	0.26	18.3%
Dummy regressor (constant)	0.32	33.7%
Dummy regressor (quantile)	0.34	36.4%

Table 3.3: Results of the trained and dummy regression models. The bold values indicate the highest scoring model per metric.

Based on these results, we trained a Random Forest regression model under the MSE loss (see Section 2.3.2) in `Pythia 0.0`. To determine the optimal set of hyperparameters for the regression model, we implemented a grid search that tests various combinations of hyperparameter values. For this grid search, we use an additional validation set, obtained by further subdividing the training set. We then perform hyperparameter optimization using this validation set. Our final difficulty predictor consists of 100 Decision Trees with a maximum depth of 10. To prevent overfitting, we set the minimum number of samples in a leaf node to 10 and the minimum number of samples required for a split to 20. Further, we train the individual Decision Trees on bootstrapped training data. We set the sample size for the bootstrapping to 75 % of the training data size.

Pythia 2.0

Since our initial publication of `Pythia 0.0`, we transitioned to using a Gradient Boosted Tree (GBT) regressor [59], as we observed an improved performance (see Section 3.3.2 below). We trained this new model using the aforementioned larger collection of 10 461 MSAs and the implementation of GBTs in the *LightGBM* Python package [89]. In analogy to `Pythia 0.0`, we trained `Pythia 2.0` under the MSE loss. We optimized the hyperparameters using the *Optuna* framework [4]. With `Pythia 0.0`, we used an 80/20 train-test-split approach for evaluation. In contrast, with `Pythia 2.0`, we evaluated the performance using a 10-fold cross validation approach (see Section 2.3.2).

3.3 Results

In the following, we first provide a thorough analysis to validate our difficulty quantification. We demonstrate that the inferred ground-truth labels accurately capture the tree search complexity under ML-based phylogenetic inference, and thus constitute an adequate training target for Pythia. Then we evaluate the prediction performance of our difficulty prediction model Pythia. We compare `Pythia 0.0` and `Pythia 2.0`, and we provide additional insights via feature importance and

Shapley value analyses. We compare the runtime of our latest model `Pythia 2.0` to the runtime of `Pythia 0.0`, demonstrating a substantial performance improvement. We additionally compare the runtime to a single ML tree inference, demonstrating the utility of Pythia as a fast-to-compute MSA assessment. Finally, we present use-cases of Pythia, including published applications, as well as an explorative analysis we performed to also predict the convergence of Bayesian phylogenetic inference methods.

3.3.1 Validating the Difficulty Quantification

Due to the lack of absolute ground-truth labels, we need to rely on the inferred difficulty labels for training Pythia. The motivation for the difficulty prediction is to limit the number of tree inferences required to sufficiently sample the tree space and obtain a representative consensus tree. To verify the label assignment for each dataset, we conducted two analyses. First, we compared the consensus tree obtained from the plausible tree set constructed from all 100 ML tree inferences (*baseline tree* T_B) to the consensus of the plausible trees we obtain when inferring only $|\mathit{difficulty} \cdot 100|$ trees (*prediction tree* T_P). Note that for this analysis we use the *difficulty* we compute according to the above definition rather than a predicted difficulty. We compare the topologies of the consensus trees using the RF-Distance. The RF-Distance between T_B and T_P is $9.6 \pm 15.8\%$ on average. This noticeable topological difference suggests that either a) the difficulty labels do not sufficiently represent the tree search behavior of the dataset, or b) 100 tree inferences do not sufficiently sample the tree space. To determine the impact of b), we repeatedly sample 99 trees out of the 100 tree inferences and compute the consensus tree T_C^i of the respective plausible tree set. We then assess the average RF-Distance between all consensus trees T_C^i. For our training data, this RF-Distance is on average $8.1 \pm 14.5\%$. We conclude that mostly b) causes the high topological distances between the *baseline tree* and the *prediction tree*. In fact, a high RF-Distance between the consensus trees T_C^i for an MSA is correlated with its difficulty. The Spearman's rank correlation coefficient is 0.88 (P-value $\ll 10^{-300}$). Thus, the more difficult the MSA, the higher the topological distances between the consensus trees T_C^i will be.

The second analysis to justify our difficulty quantification ensures that selecting the number of tree inferences based on the difficulty does not negatively impact the quality of the tree inference. In general, the difficulty cannot predict the number of tree searches required to sufficiently sample the tree space, as this number also depends on the implemented tree inference heuristic. However, since we define the difficulty based on 100 ML tree inferences in RAxML-NG, we can use the difficulty to approximate the number of required tree inferences, when again using RAxML-NG, as a fraction of 100. Thus, to analyze the influence of the difficulty on the quality of the tree inference, we compare the log-likelihoods obtained from 100 independent RAxML-NG tree searches ($LnLs_{100}$) to the log-likelihoods of $|\mathit{difficulty} \cdot 100|$ tree searches ($LnLs_{\text{diff}}$) for all MSAs in our training data. We compare the respective best found log-likelihoods LnL^*_{100} and LnL^*_{diff}, as well as the average log-likelihoods $\overline{LnL}_{100}$ and $\overline{LnL}_{\text{diff}}$.

For 81% of the MSAs, the best found log-likelihoods LnL^*_{100} and LnL^*_{diff} are identical. For the remaining 19% of the MSAs, LnL^*_{diff} is on average $\ll 0.01\%$ worse than LnL^*_{100}. The average log-likelihoods $\overline{LnL}_{100}$ and $\overline{LnL}_{\text{diff}}$ deviate on average by 0.01% only.

This analysis only serves for justifying the definition of our difficulty quantification. Predicting the number of tree inferences as a fraction of 100 is only applicable to ML tree inference with RAxML-NG. It should further be mentioned, that RAxML-NG only infers 20 trees by default. Thus, simply increasing the number of tree inferences to $\lfloor \mathit{difficulty} \cdot 100 \rfloor$ is discouraged.

Given these analyses, we conclude that our difficulty quantification is sufficiently accurate to capture the tree search complexity and the behavior of an MSA under ML based phylogenetic analysis. Consequently, the difficulty quantification can serve as a prediction target for training machine learning models.

3.3.2 Prediction Performance Evaluation

Pythia predicts the difficulty of an MSA on a scale of 0.0 (easy) to 1.0 (difficult). Our initial model, `Pythia 0.0`, has an MAE of 0.09 and an MAPE of 2.9%. With `Pythia 2.0`, we slightly improved the prediction accuracy to an MAE of 0.08 and an MAPE of 2.2%. When analyzing the prediction error, we noticed that Pythia tends to overestimate the difficulty of MSAs with a difficulty ≤ 0.3 and to underestimate the difficulty for MSAs with a difficulty > 0.3 (see Figure 3.4). We observe this effect for both, `Pythia 0.0`, and `Pythia 2.0`. However, with `Pythia 2.0`, this effect is less pronounced. We suspect that this constitutes an artifact of the uneven difficulty distribution in the training data (see Figure 3.3). Thus, future work should focus on obtaining more intermediate and difficult MSAs.

We decided to train a single predictor for DNA, AA, and morphological data rather than training a separate predictor for each data type. To justify this design choice, we assess the prediction error of `Pythia 2.0` per data type separately. As Figure 3.5 shows, the overall performance of Pythia is comparable over all three data types.

To further justify this design choice, we also trained three separate predictors per data type ($f_{\text{DNA}}, f_{\text{AA}}, f_{\text{M}}$), and one predictor using all three data types simultaneously (f_{all}). To ensure a fair comparison, we split the training data of 10 461 MSAs into a *training set* and a *test set*. The *test set* is only used for performance comparison. Note that we trained a new predictor for this experiment instead of using `Pythia 2.0` to ensure that the predictor was trained on the *training set* only for a fair comparison. The *training set* comprises 8368 MSAs, and the *test set* contains the remaining 2093 MSAs. We split the data using stratified sampling such that the respective proportion of DNA, AA, and morphological MSAs is preserved. Thus, f_{DNA} is trained on 7287 DNA MSAs, f_{AA} on 766 AA MSAs, and f_{M} on 315 morphological MSAs. The predictor f_{all} is trained on the full *training set* of 8368 MSAs (DNA, AA, and morphological). We then compare the MAE of the per-data-type prediction models to the prediction performance of f_{all} using the MSAs of the respective data type in the *test set*. For DNA and AA MSAs, the prediction accuracy

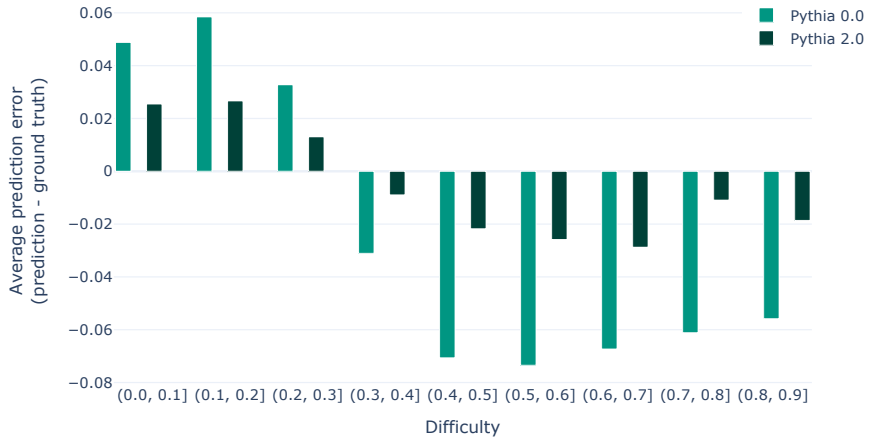


Figure 3.4: Average prediction error per difficulty range. The figure shows the error for `Pythia 0.0` and `Pythia 2.0` on their respective training datasets. We compute the prediction error as *predicted difficulty - ground-truth difficulty*.

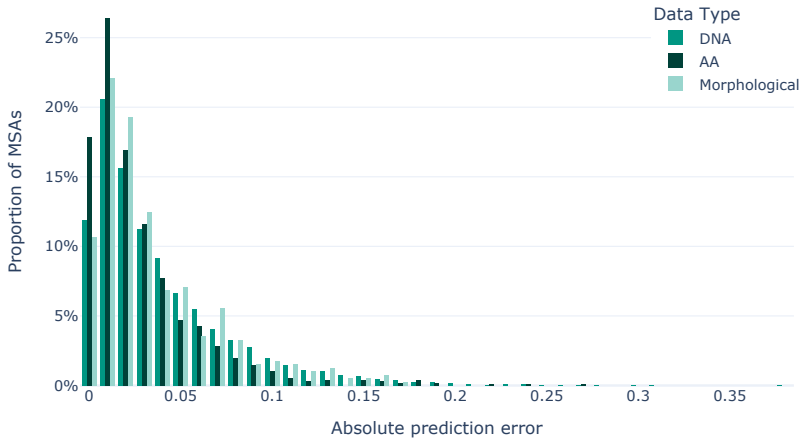


Figure 3.5: Absolute prediction error of `Pythia 2.0` per data type.

is identical when only using DNA/AA MSAs for training, compared to using all three data types (MAE = 0.1). For morphological data, we observe a slight decrease in MAE for f_{M} (MAE = 0.12) compared to f_{all} (MAE = 0.11). We suspect that this is caused by the lack of sufficient training data. The predictor trained exclusively on morphological MSAs was trained on only 315 MSAs yielding it more challenging for the predictor to generalize to unseen data. In contrast, the predictor using MSAs of

Feature	`Pythia 0.0`	`Pythia 2.0`
Patterns-over-taxa	8.8%	**39.6%**
RF_{MP}	20.8%	31.5%
Bollback Multinomial	1.7%	4.8%
N^*_{MP}	**54.7%**	4.7%
Entropy	10.0%	3.8%
% Gaps	2.0%	3.7%
% Invariant	0.8%	3.2%
Patterns-over-sites	-	3.0%
Sites-over-taxa	1.2%	2.9%
Pattern Entropy	-	2.8%

Table 3.4: Feature importances in percent for `Pythia 0.0` and `Pythia 2.0`. We computed the feature importances using the gain-based feature importance.

all data types can detect patterns that are independent of the underlying MSA data type, and can thus learn more general difficulty characteristics on a substantially larger MSA collection.

3.3.3 Feature Importance and Model Insights

Table 3.4 shows the feature importance for `Pythia 0.0` and `Pythia 2.0`, respectively. We computed the gain-based feature importance that directly measures the contribution of each feature to the performance improvement during training. `Pythia 0.0` heavily relied on the MP-based features RF_{MP} and N^*_{MP} with a combined feature importance of approximately 75%. While `Pythia 2.0` still heavily relies on the RF_{MP} feature (31.5%), the most important feature is the patterns-over-taxa ratio, with approximately 40%. We suspect that `Pythia 2.0` relies less on the N^*_{MP} feature (4.7%), since the new training data shows an abundance of MSAs where $N^*_{\text{MP}} = 1$, indicating that all 24 inferred MP trees have distinct tree topologies (see Figure 3.3).

Using *Shapley values* [157] (see Section 2.3.5), we can gain additional insights into the contribution of individual features to the predicted difficulty. `Pythia 2.0` includes a command line option, as well as a dedicated method to plot the Shapley values for a given MSA to allow for interpretable predictions, and to support Pythia users in gaining additional insights. For these plots, as well as for the following analyses, we rely on the *SHAP* Python package [112].

Using the Shapley values computed on multiple MSAs, we can visualize a feature value contribution trend for the overall prediction via a so-called *beeswarm* plot. A beeswarm plot simply combines multiple, individual Shapley value plots into a single figure. Figure 3.6 shows the beeswarm plot of `Pythia 2.0` for all 10 461 MSAs in our training data. The features are sorted by importance, with the topmost feature being the most important one. Note that the feature order differs compared to the feature importances stated in Table 3.4, since the feature importance is computed differently. The color of each dot corresponds to the magnitude of the respective

feature value, with darker colors corresponding to higher values. The x-axis reflects the contribution of a feature value to the model output (the predicted difficulty). Negative values indicate a reduction in predicted difficulty, while positive values indicate a difficulty increase. The x-axis is relative to the *baseline* prediction. In the case of a GBT regressor trained via the MSE loss (see Section 2.3.2), this baseline prediction simply corresponds to the average difficulty in the training data.

The beeswarm plot clearly shows that a higher signal in the MSA corresponds to lower predicted difficulties. This is predominantly visible for the patterns-over-taxa ratio, with contributions being as low as -0.3. This trend is also visible for the patterns-over-sites ratio, the sites-over-taxa ratio, and the proportion of gaps. Analogously, a higher information content (as captured by the Entropy, the Bollback Multinomial, and the Pattern Entropy) has a negative impact on the prediction, thus contributing towards an MSA being easier to analyze. Note that, while *higher* values indicate a *higher* information content for the Entropy and the Bollback Multinomial, *lower* values indicate a *higher* information content for the Pattern Entropy. Interestingly, a *higher* proportion of invariant sites contributes *negatively* to the predicted difficulty and vice versa. This is unexpected, since a high proportion of invariant sites is expected to induce less variation (and thus signal) for phylogenetic inference. Consequently, we would expect higher proportions of invariant sites to induce higher predicted difficulties. Further investigating this counter-intuitive trend will be the subject of future work. As expected, higher RF_{MP} values contribute to an MSA being predicted as more difficult, while lower values contribute to predicting lower difficulties. This trend is expected, as our difficulty definition under ML reflects the ruggedness of the tree space and correlates well with the ruggedness under MP. Analogously, a higher proportion of unique topologies (N^*_{MP}) tends to contribute positively to the predicted difficulty. However, this trend is not as clearly pronounced. We again suspect that this is caused by the high abundance of MSAs in the training data where $N^*_{\mathrm{MP}} = 1$.

3.3.4 Runtime Evaluation

In the following, we benchmark the runtime of `Pythia 2.0` against `Pythia 0.0`, and against ML tree inference using RAxML-NG. Additionally, we provide an overview of the runtime contributions of MSA parsing, feature computation, and difficulty prediction to the overall difficulty prediction runtime for a single MSA in `Pythia 2.0`. We used all 10 461 MSAs in our training dataset for all subsequent analyses.

`Pythia 2.0` is on average $2.4 \pm 1.2\times$ faster than `Pythia 0.0` ($2.0\times$ median). Multiple factors contribute to this speedup. Firstly, we reduced the number of MP trees in `Pythia 2.0` to 24, compared to the 100 MP trees in `Pythia 0.0`. We also refactored the internal representation of MSAs in `Pythia 2.0`. This allows us to natively compute the number of patterns, the proportion of invariant sites, and the proportion of gaps in Python using our MSA representation. In contrast, in `Pythia 0.0`, we relied on RAxML-NG for computing these features. While the RAxML-NG-based implementation is faster, including these features as MSA properties in `Pythia 2.0` (and thus removing the RAxML-NG dependency) improves the re-usability of our

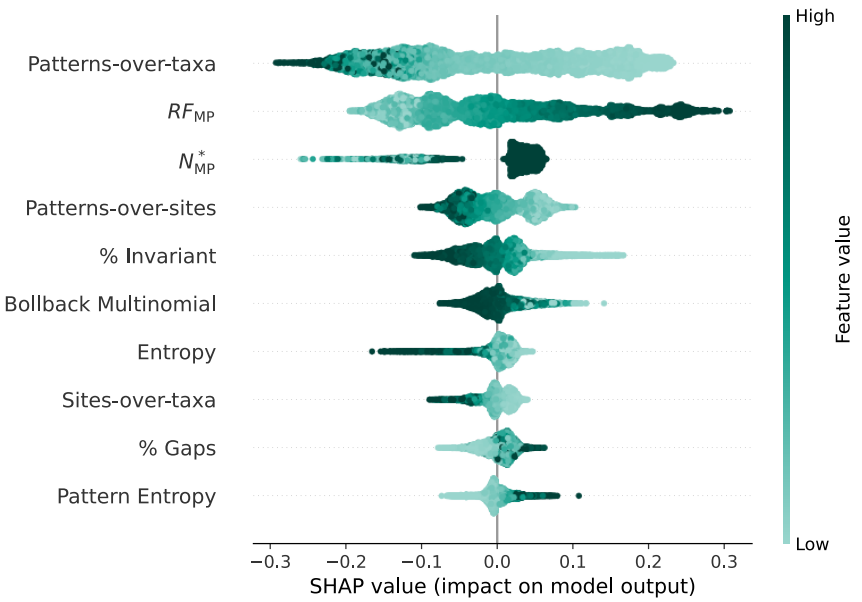


Figure 3.6: Beeswarm plot of Shapley values of `Pythia 2.0` for the 10 461 MSAs in our training data.

MSA representation in our PyPythia Python library for other applications that rely on MSA attributes. We observe that the speedup of `Pythia 2.0` compared to `Pythia 0.0` increases with increasing MSA size (number of taxa times number of patterns in the MSA; see Figure 3.7). For instance, the average speedup for MSAs with a size that exceeds 100 000 is 4.2× (4.1× median).

The main goal of using Pythia is to gain insights into the expected behavior of the MSA under ML phylogenetic inference. This allows for a more informed analysis setup and adjustment of expectations before conducting any time- and resource-intensive analyses. To demonstrate the utility of Pythia, we thus compare the runtime of predicting the difficulty for an MSA using `Pythia 2.0` to the runtime of performing a *single* ML tree inference using RAxML-NG. Across all MSAs, we observe an average speedup of 21.5±43.8× (7.7× median). We observe a high standard deviation since the speedup predominantly depends on the MSA size. Figure 3.8 shows the speedup as a function of MSA size. For (very) small MSAs with a size below 1000, `Pythia 2.0` is slightly slower than a single ML tree inference on average. However, the average speedup on MSAs with a size exceeding 100 000 is above 50× (37× median). Note that an MSA with 100 taxa and 1000 patterns already falls into this category.

Finally, we analyzed the contribution of various computational steps required for difficulty prediction in `Pythia 2.0` with respect to the overall runtime. Figure 3.9 shows the average contribution across all MSAs in our training data. Note that we summarized the feature computation into three separate groups: the *MP-based* group (RF_{MP} and N^*_{MP}), the *Information Content* group (Entropy, Bollback Multinomial, Pattern Entropy), and the *MSA Attributes* group (sites-over-taxa, patterns-

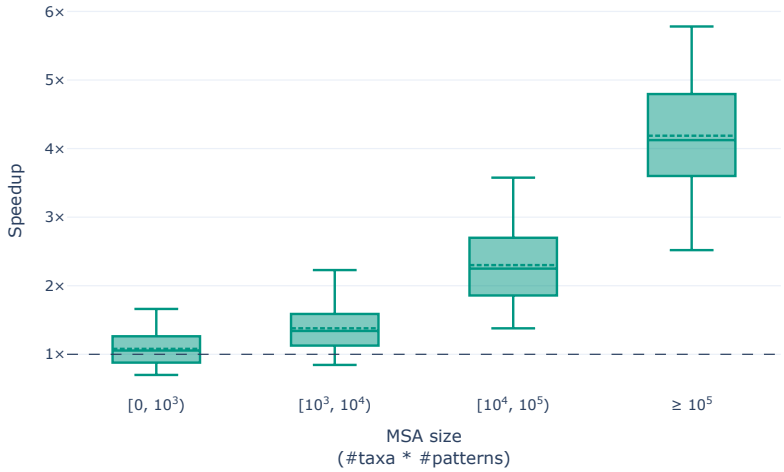


Figure 3.7: Speedup of `Pythia 2.0` compared to `Pythia 0.0` as a function of MSA size. MSA size is computed as the number of taxa times the number of unique site patterns in the MSA. To improve visualization, we only show data between the 5th and 95th percentile.

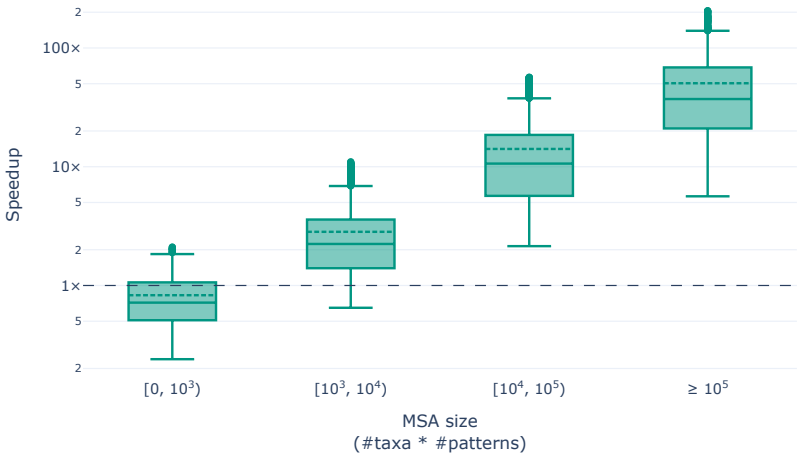


Figure 3.8: Speedup of `Pythia 2.0` compared to a single ML tree inference using RAxML-NG as a function of MSA size. The MSA size is computed as the number of taxa times the number of unique site patterns in the MSA. Note that the y-axis is logarithmic. To improve visualization, we only show data between the 5th and 95th percentile.

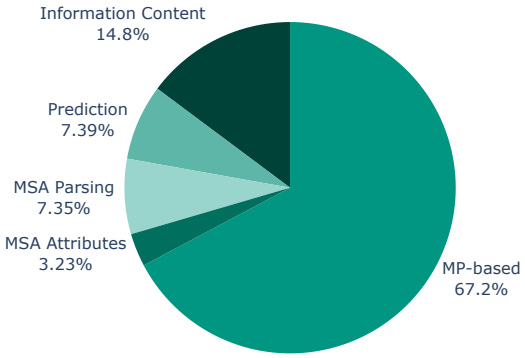


Figure 3.9: Contribution of computational steps in `Pythia 2.0` to predict the difficulty of an MSA. The pie chart shows the average contribution across the 10 461 MSAs in our training dataset.

over-taxa, patterns-over-sites ratios, % invariant, % gaps). *MSA Parsing* captures the runtime for reading the MSA from file and transforming it into our internal MSA representation. *Prediction* refers to loading the pretrained Pythia prediction model and performing the final difficulty prediction. The runtime is dominated by the computation of the MP-based features, with an average contribution of 67%. This indicates that future work in Pythia should focus on improving the runtime of the MP tree inference, either by improving the MP implementation in RAxML-NG (which is developed in our research group) or by replacing RAxML-NG with a faster alternative.

3.3.5 Applications of Pythia

We suggest predicting the difficulty using Pythia *before* conducting any ML phylogenetic inference, as this allows for more targeted analysis strategies. For example, for a difficult MSA, the user should be careful to report a single ML tree as the best-known tree, since the tree space most likely exhibits multiple, indistinguishable local optima. The user should also be aware that a difficult MSA requires more independent tree searches to construct a reliable consensus tree than an easy MSA. Furthermore, difficult MSAs require a more careful consideration of necessary additional phylogenetic analyses and post-processing steps. Especially for very difficult MSAs (difficulty > 0.8) we suggest considering improving upon the difficulty of the MSA before the analysis. This is because a phylogenetic analysis on very difficult MSAs, will most likely not yield a well-resolved tree, even if a consensus of numerous almost equally likely, yet topologically distinct ML trees is built.

Ever since our initial publication in 2022, Pythia has been used in multiple studies to analyze and characterize collections of MSAs [77, 99, 144, 170, 176], or to select data based on the predicted difficulty [17, 30, 95]. Additionally, Pythia proved to be a useful prediction feature in multiple novel machine-learning based applications. Collienne *et al.* [33] showed that Pythia is the dominant prediction feature when

predicting the *instability* of taxon placement into an existing phylogenetic tree. Instability means that the evolutionary relationship of the taxa in the pre-existing tree changes upon placement or addition of a new taxon. Furthermore, Ecker *et al.* [37] recently proposed novel machine learning-based bootstrap support values for bipartitions in phylogenetic trees, including Pythia as a prediction feature. Similarly, Wiegert *et al.* [188] published the *Educated Bootstrap Guesser* (EBG), a machine learning-based framework that accurately predicts Felsenstein bootstrap support values for a given phylogeny. EBG does not rely on Pythia as a prediction feature, but instead heavily relies on a fast-to-compute approximation of the Felsenstein bootstrap using MP trees. The idea for using MP-based features in EBG stems from our analysis of Pythia's feature importances. In our work, we showed that MP-based tree inferences are highly informative in approximating ML tree inference behavior while being substantially faster.

Studying the influence of the number of starting trees in ML phylogenetic inference across a wide range of empirical and simulated MSAs, Liu *et al.* [108] found that Pythia can roughly approximate the number of required individual tree inferences to sufficiently sample the ML tree space.

Building on Pythia, and on a thorough performance comparisons of various ML tree inference heuristics by Hoehler *et al.* [77], Togkousidis *et al.* [175] implemented an adaptive ML tree inference procedure for the popular RAxML-NG ML tree inference tool (*adaptive RAxML-NG*). Adaptive RAxML-NG predicts the difficulty of an MSA using Pythia and subsequently categorizes the MSA into *easy* (difficulty ranging from 0.0 to 0.3), *intermediate* (0.3 to 0.7), and *difficult* (> 0.7). Depending on the difficulty category, adaptive RAxML-NG adjusts the default number of inferred trees, and also executes an appropriately adapted tree search strategy. Compared to *standard* RAxML-NG by Kozlov *et al.* [96], adaptive RAxML-NG exhibits an average speedup of 16× on easy and difficult MSAs, and of 1.8× on intermediate MSAs. Note that the speedup cannot exclusively be attributed to the difficulty-induced changes, as Togkousidis *et al.* [175] also adjusted the difficulty-independent tree inference algorithm to allow for faster tree inferences. Yet, adaptive RAxML-NG demonstrates the substantial potential of integrating Pythia-informed strategies into phylogenetic analyses pipelines.

3.3.5.1 Exploration of MCMC Convergence Prediction

In our work, we focused on predicting the difficulty of ML phylogenetic inferences. Another popular method to explore the tree space of an MSA is Markov chain Monte Carlo (MCMC) based Bayesian phylogenetic inference. Since both methods, ML and MCMC, rely on the same input MSA and on the same likelihood function, we suspect the difficulty to also be reflected in the apparent convergence speed of MCMC methods. Since MCMC phylogenetic analyses constitute a time- and resource-intensive task, we only explored this potential correlation using three exemplary MSAs. A thorough exploration of the connection to difficulty prediction is beyond the scope of this work.

The features we use to predict the difficulty of an MSA are independent of the inference method used for the subsequent analyses. However, as we describe in Section 3.2.1, our difficulty quantification is based on 100 tree inferences using RAxML-NG, which implements the ML method. Therefore, our predictions might be biased towards ML analyses and potentially do not describe the ruggedness of the tree space in a model-independent manner. To assess if our predictions can be generalized, we compare our difficulty prediction to convergence diagnostics of MCMC based phylogenetic analyses. For three DNA MSAs (D27 [74], D125 [139], and D354 [63]) we performed MCMC analysis using MrBayes [146]. We ran four chains for 10 million generations each using the general time reversible (GTR) model with four Γ rate categories to account for among site rate heterogeneity. MrBayes reports the average standard deviation of split frequencies (ASDSF; split frequencies: relative number of occurrence of splits/bipartitions in the set of posterior trees) as a convergence diagnostic metric and suggests executing additional generations as long as the ASDSF is ≥ 0.01. D125 is an easy dataset with an expected clear, single likelihood peak. The difficulty according to our definition is low ($\ll 0.1$) and MrBayes appeared to converge: the ASDSF value dropped below 0.01 after 150 000 generations and is $\ll 0.01$ after only 1 million generations. D27 exhibits at least two distinct likelihood peaks, suggesting that the MSA is rather difficult to analyze [100]. The difficulty according to our definition is 0.45 and after 10 million generations, MrBayes reported an ASDSF of 0.011, indicating that the MCMC did not converge to a single local optimum. D354 exhibits a rugged likelihood surface [63], so we expect a high difficulty and no convergence. The assigned difficulty for D354 is 0.6 and after 10 million generations the ASDSF was 0.009. According to MrBayes this suggests convergence and adding more generations should improve the ASDSF. However, we observed that the ASDSF did not improve during the last 2 million generations, and adding more generations did not further improve the ASDSF. D125 with 125 taxa and approximately 30 000 sites is a larger MSA than D354 with 354 taxa and only 460 sites. Yet, D125 apparently converged after 1 million generations, while for D354 the ASDSF dropped below 0.01 only after 8 million generations. The smallest MSA D27 with 27 taxa and 1940 sites indicated no convergence after 10 million generations, according to the ASDSF. We thus suspect that the number of generations required for the MCMC is correlated to the difficulty rather than to the size of the dataset. A thorough analysis of this potential correlation remains the subject of future work.

3.4 Discussion

Predicting the difficulty of MSAs to gain *a priori* insights into the expected behavior of phylogenetic tree searches and the shape of the likelihood surface constitutes a vital step towards faster phylogenetic inference and a more targeted setup of the computational analyses and post-analyses. Our difficulty prediction allows for careful consideration of the number of tree inference required to sufficiently sample tree space *prior* to ML analyses. Especially for easy MSAs, this has the potential to save valuable time and resources. In this paper, we presented a quantifiable definition of difficulty for MSAs and showed that this definition adequately represents the

ruggedness of the tree space of the MSA under ML. Using this definition, we trained Pythia, a machine-learning based prediction model that predicts the difficulty on a scale ranging between 0.0 to 1.0. Our initial version `Pythia 0.0` achieved high prediction accuracy with an MAE of 0.09 (MAPE 2.9%). We trained our latest model, `Pythia 2.0`, on approximately three times more MSAs and slightly improved the prediction accuracy to an MAE of 0.08 (MAPE 2.2%). We showed that we can accurately predict the difficulty for DNA, AA, and morphological MSAs, making Pythia applicable to a broad range of phylogenetic analyses. Using 10 fast-to-compute prediction features, `Pythia 2.0` is on average approximately 20 times faster than a *single* RAxML-NG ML tree inference. The more taxa and sites the MSA has, the faster the feature computation is relative to a single ML tree inference, making Pythia especially valuable for phylogenetic analyses on MSAs with many sites and taxa.

We conclude that predicting the difficulty of an MSA before any tree inference allows for faster analyses, adjusting user expectations regarding the stability of the inferred tree, and that Pythia should be included in ML phylogenetic inference pipelines by default.

4. Simulations of Sequence Evolution: How (Un)realistic They Are and Why

This chapter is derived from the peer-reviewed open-access publication:

> Johanna Trost, **Julia Haag**, Dimitri Höhler, Laurent Jacob, Alexandros Stamatakis and Bastien Boussau. "Simulations of Sequence Evolution: How (Un)realistic They Are and Why." *Molecular Biology And Evolution*, Volume 41, Issue 1, January 2024. https://doi.org/10.1093/molbev/msad277

This paper was a collaboration between our research group with Johanna Trost (Biometry and Evolutionary Biology Laboratory (LBBE), University Claude Bernard, Lyon, France) and her supervisors Laurent Jacob (Laboratory of Computational and Quantitative Biology (LCQB), Sorbonne Université, Paris, France) and Bastien Boussau (Biometry and Evolutionary Biology Laboratory (LBBE), University Claude Bernard, Lyon, France). Johanna Trost (JT) and our lab members Dimitri Höhler (DH) and Julia Haag (JH) contributed equally to this publication.

JT and DH simulated the data collections. JT trained and evaluated the CNN classifiers, including all CNN related feature importance analyses. JH trained and evaluated the GBT classifiers. JH desgined and implemented the required features for the GBTs, with DH contributing the *randomness* features. JH performed all feature importance analyses related to the GBTs, as well as the analyses of the relationship between the Pythia difficulty and the classification accuracy.

4.1 Background and Motivation

Reconstructing the evolutionary history of species or genes by inferring phylogenetic trees is a ubiquitous task in comparative genomics. Typically, phylogenetic inference is based on an MSA that contains aligned sequences of the species under study (see Section 2.1.2). A plethora of inference algorithms, tools, and models have been developed to infer phylogenetic trees based on the MSA, for example RAxML-NG [96], IQ-TREE [121], BEAST [22], or RevBayes [80]. When developing novel methods and validating their performance, comparing them to existing state-of-the-art methods on both, empirical, and simulated data is mandatory. Simulated data are particularly useful for conducting inference accuracy and implementation verification assessments, when a known, ground truth phylogeny is required. Both, simulation tools [28, 54, 114], and state-of-the-art inference methods are based on probabilistic models of sequence evolution. Most of the latter exploit models through likelihood functions, by searching for trees that maximize this likelihood [96, 121] or by sampling from posterior distributions via Metropolis-Coupled Markov Chains, which also rely on likelihood computations [22, 80]. Alternatively, researchers have started to explore likelihood-free approaches (for examples outside our field, see Lueckmann *et al.* [111]). These approaches sample the posterior density instead of evaluating it, and thereby avoid computing the likelihood. The resulting simulated samples are used to build an estimate of the posterior distribution. This so-called simulation-based inference paradigm was pioneered in population genetics under the Approximate Bayesian Inference (ABC) framework [34], and extended over the past decade to neural density estimation techniques [131], where a neural network is trained to output the correct distribution of parameters for a given, observed input. In the context of phylogenetic inference, neural density estimation has been restricted to the reconstruction of a single tree rather than a full distribution. For example, Suvorov *et al.* [172] use convolutional neural networks to reconstruct phylogenies from MSAs with four sequences, and Nesterenko *et al.* [129] use a transformer-based network architecture to predict evolutionary distances between all pairs of sequences in an MSA.

In all of the above contexts – evaluation, likelihood-based, or -free inference – it is essential that the probabilistic model of sequence evolution is consistent with empirical data. For evaluation, performance on simulated data is indicative of performance on empirical data, only if the two are sufficiently similar. For inference, a misspecified model can induce inaccurate and misleading results. For training machine learning-based methods, it is important that the training data and empirical data are sufficiently similar to circumvent "out-of-distribution" problems [32]. Such problems occur when the training data does not accurately represent the empirical data, or when it misses subgroups of the empirical data: the trained method has never "seen" data similar to the empirical data, and can thus behave poorly.

Authors using simulated data in their publications typically set simulation parameters according to attributes (for example MSA lengths, or proportions of gaps) of empirical reference MSAs (see for example Price *et al.* [141]). Some also attempt to extract or sample simulation parameters from ML estimates in large scale empirical

databases, such as TreeBASE [138]. The intention is that, thereby, simulated data will more closely resemble empirical data [1, 77]. Despite this effort, there still exist performance and/or program behavior differences on simulated versus empirical data. For example, Guindon *et al.* [64] conclude that comparing methods using simulated data is not sufficient, as "the likelihood landscape tends to be smoother than with real data", and Hoehler *et al.* [77] observe differences between empirical and simulated data when comparing ML phylogenetic inference methods. They conclude that there exist not yet understood differences between simulated and empirical data.

Here, we introduce a metric to quantify how realistic a substitution model is, by simulating data using the respective model, and training a classifier to discriminate between simulated and empirical data. We expect realistic models to produce simulated data that are difficult to discriminate against empirical data and induce low classifier accuracy. We leverage recent data simulation tools [28, 54, 114] that are feature-rich and support a wide range of evolutionary models and simulation parameters. We show that we can distinguish simulated from empirical data with up to 99% classification accuracy, depending on the used simulation model. We present two different and independently developed machine learning approaches exploiting distinct MSA characteristics for this classification task: One, using Gradient Boosted Trees (GBT), and another approach based on a Convolutional Neural Network (CNN). We show that prediction accuracy decreases, the more complex the model of evolution used in simulations becomes. Yet, we also observe exceptions to this general trend. For the most complex models in our experimental setup, the prediction accuracy is still very high, with the CNN-based classifier achieving prediction accuracies ≥ 0.93 on all tested models. This indicates that simulated MSAs are easy to distinguish from empirical MSAs, as they do not appear to reproduce some characteristic features of empirical MSAs. We further show that simulating indels remains a challenging task, as including indels results in higher classification accuracies with the CNN classifiers compared to simulations without indels. Further, based on the feature importances of the GBT classifiers, we show that simulated data exhibit more evenly distributed site substitution patterns than empirical data.

4.2 Methods

The goal of our study was to be able to distinguish between empirical and simulated DNA and AA data with high accuracy under increasingly complex models of sequence evolution. Figure 4.1 depicts our experimental setup for one exemplary set of empirical MSAs (*empirical data collection*) and one exemplary model of evolution. Using the empirical data collection and the given model of evolution, we simulated a new set of MSAs (*simulated data collection*) using the AliSim sequence simulator [114]. Based on the empirical and simulated data collections, we completely independently trained two distinct classifiers for each simulated data collection: a Gradient Boosted Tree (GBT) and a Convolutional Neural Network (CNN).

In the following sections, we describe our experimental setup, the sequence simulation process, and both classification methods.

Please note that this study was a collaborative research project. My main contribution to this study was the training and evaluation of the GBT classifiers, with Dimitri Höhler contributing the *randomness* features. Johanna Trost and Dimitri Höhler conducted the MSA simulations, and Johanna Trost trained and evaluated the CNN classifiers. Thus, the following section only briefly summarizes MSA simulations and the training of the CNN classifiers. For further details, we refer the interested reader to our paper *Simulations of Sequence Evolution: How (Un)realistic They Are and Why* [178].

4.2.1 MSA Simulations

For our study, we separately considered DNA and AA data. We simulated 15 sets of MSAs, seven sets containing DNA MSAs and eight containing AA MSAs, respectively. In the following, we refer to an MSA set as a *data collection*. To simulate the MSAs for each data collection, as well as for data discrimination, we used two empirical data collections as reference, one per data type. The empirical DNA data collection contains MSAs obtained from TreeBASE [138]. The empirical AA data collection consists of MSAs obtained from the HOGENOM database [136]. See Section 2.1.2.1 for further details on both databases. We removed outliers based on MSA length (number of sites), number of sequences, as well as MSAs with less than four sequences to allow for a reliable and efficient analysis. Very long sequences would inflate the memory footprint of the CNN, while very short MSAs are uncommon. Removing outliers allowed us to deploy a balanced and representative data collection that facilitates robust and unbiased predictions.

Moreover, empirical MSAs may contain sites with ambiguous AA/DNA states (see Section 2.1.1). As a further pre-processing step, yet applied exclusively for the CNN classifier, we removed all MSA sites containing at least one ambiguous character, as they would bias the prediction. For AA data this concerned 912 out of 6969 MSAs, and we removed 1.34% of all sites within these 912 MSAs. Furthermore, 13.24% of sites in 6117 DNA MSAs were removed.

For each data type, we generated simulated data collections based on the corresponding empirical data collection, resulting in identical numbers of simulated and empirical MSAs. We simulated data using the AliSim sequence simulator [114] under several evolutionary models ranging from easy to complex, in terms of number of free parameters and computational methods used to derive respective AA substitution models. See Section 2.1.3.2 for details on models of sequence evolution. The goal of this setup was to progressively increase simulation realism. First, we simulated five DNA and seven AA data collections without gaps, which allowed us to characterize the realism of substitution models per se. To this end, we removed all sites containing gaps (-) from all empirical MSAs. The resulting empirical data collections contain 7637 DNA MSAs and 6971 AA MSAs, respectively. We henceforth refer to these data collections as *gapless* data collections. Second, we simulated two DNA and one AA data collection with indel events, based on the empirical MSAs containing gaps (9460 DNA MSAs and 6971 AA MSAs).

In the following, we describe the simulation procedures for both data types and our approach to simulate indel events in more detail.

4.2.1.1 DNA Simulation

We simulated seven DNA data collections in total (5 gapless and 2 with simulated indel events), with each data collection being simulated under a different evolutionary model with increasing model complexity. We used the following models of evolution. As the simplest model, we used the Jukes-Cantor (*JC*) model (equal substitution rates and equal base frequencies) [85]. We also used the *HKY* model (four degrees of freedom) [70], and the General Time Reversible (*GTR*) model (eight degrees of freedom) [173]. To account for among site rate heterogeneity, we additionally simulated under GTR in conjunction with the Γ model of rate heterogeneity [192] using four discrete rates (*GTR+G*). The most complex model of evolution we used for simulation was the GTR+G model, with an additional free parameter to accommodate the proportion of invariant sites (*GTR+G+I*) [161].

We selected 9460 empirical MSAs (*Set1*) from TreeBASE [138, 183] as the basis for our simulations. We removed all sites containing gaps (`-`) or fully undetermined characters (`N`) from the MSAs of *Set1*. Thereby, we obtained 7637 non-empty MSAs (that is, MSAs that still contained at least one site), which we defined as *Set2*. This induced an MSA length reduction of around 55% compared to *Set1*. We based our five simulated DNA data collections without indel events on *Set2*, and the two data collections with indels on *Set1*.

AliSim simulates sequences along a given phylogenetic tree (see Section 2.1.5). We avoided the problem of simulating realistic phylogenetic trees for this purpose by initially estimating a best-known ML tree using RAxML-NG [96] (default parameters), for every MSA of *Set2* under each of the five evolutionary models (JC, HKY, GTR, GTR+G, GTR+G+I). We then used the inferred phylogeny and respective estimated model parameters to simulate MSAs using AliSim [114] based on every MSA of *Set2*, without specifying an indel model. In the following analyses, we refer to the resulting five gapless data collections as JC, HKY, GTR, GTR+G, and GTR+G+I according to the model of evolution used to simulate the respective data collection. In Section 4.2.1.3 below, we describe the simulation of the two additional DNA data collections with indel events.

4.2.1.2 AA Simulation

We simulated seven AA data collections limited to substitution events only, and one additional data collection *with* indels. The most rudimentary evolutionary model we used is the *Poisson* model, with equal substitution rates and equal equilibrium frequencies. We further used two empirical substitution models: the *WAG* [186] and the *LG* [103] model. The LG model is expected to produce more realistic simulations than the WAG model as the former was derived from a larger and more diverse data collection, using more refined inference techniques than the latter. These substitution models use a single set of equilibrium frequencies (one AA profile) to simulate all sites in an MSA. We also used mixture models that incorporate heterogeneity

among sites by employing multiple profiles. In such models, a profile is drawn from a set of profiles to simulate a single site. We used the following two AA mixture models: the C60 model with 60 profiles (*LG+C60*) [162] and the more recent UDM model with 256 profiles (*LG+S256*) [153]. The advantage of the latter model is that each profile is assigned a probability of generating a site, while under the C60 model, profiles are drawn with equal probabilities. In addition, the UDM model is based on a subset of MSAs from the HOGENOM database, and should therefore generate MSAs that are similar to empirical HOGENOM MSAs. To increase model complexity, we performed further simulations accounting for among site heterogeneity using the Γ model [192], in analogy to the DNA simulations. We simulated two data collections, one using four discrete Γ rate categories (*LG+S256+G4*) and the second one by applying a continuous Γ distribution (*LG+S256+GC*).

In analogy to the simulated DNA data collections, we refer to the simulated AA data collections according to the model of evolution used. The gapless AA data collections are Poisson, WAG, LG, LG+C60, LG+S256, LG+S256+G4, and LG+S256+GC. In the following section, we describe the simulation procedure for the data collection with indels.

4.2.1.3 Simulating Indels

In addition to the gapless data collections, we simulated two DNA, and one AA data collection *with* indels. For both data types, we used the most complex models of evolution as a basis (GTR+G+I for DNA, LG+S256+GC for AA).

To generate the first DNA data collection with indels, we performed tree inferences using RAxML-NG under the GTR+G+I model for each MSA of DNA *Set1*. We then simulated MSAs with indels using two distinct procedures to generate two distinct data collections. For the first data collection, we simulated the MSAs in the same way as for the gapless collections. Then, we superimposed the gap pattern of the MSAs used as the basis of the simulation onto the simulated MSAs. We refer to this procedure as the *mimick* procedure and denote the resulting data collection as GTR+G+I+mimick.

For the second data collection, as well as the AA data collection with indels, we simulated the MSAs using not only the inferred trees and estimated evolutionary model parameters, but also by specifying indel parameters. In the following, we describe the procedure to infer and validate these parameters. We performed this procedure for both DNA and AA data collections separately. We refer to this procedure as the *sparta* procedure. We first used the SpartaABC tool [110] to obtain indel-specific parameters from a subset of empirical MSAs. Here, we employed the rich indel model (RIM), which differentiates between insertion and deletion events using five free parameters. The inferred parameters are: Insertion and deletion rate (I_R, D_R), root length (RL), and the parameter A that controls the Zipfian distribution of insertion and deletion lengths (A_I, A_D). We will henceforth refer to this set of parameters as *empirical indel parameters*.

To simulate MSAs, we drew indel parameters from the joint parameter distribution of empirical indel parameters. To approximate the probability density function

(PDF), we applied Gaussian kernels to the five principal components of the indel parameters. This choice was based on our observation that a more accurate match is achieved between the empirical parameters' empirical cumulative distribution function (ECDF) and the resulting parameters' ECDF when using the principal components. For the Gaussian kernels, we determined the bandwidth using Scott's rule of thumb [155]. Moreover, we employed the kernel-density estimation implementation by Virtanen *et al.* [182], although it tends to overestimate the distribution's actual edges. To mitigate this issue, we re-sampled values if they exceeded the limits of the parameter prior limits chosen by Loewenthal *et al.* [110]. To validate our approach, we compared the ECDF of the empirical parameter values with the ECDF of parameters sampled from the empirical PDF for each indel parameter type. We denote the resulting DNA data collection as GTR+G+I+sparta, and the resulting AA data collection as LG+S256+GC+sparta.

4.2.2 Classification Methods

To distinguish simulated from empirical MSAs, we developed two distinct approaches. One approach is a classical machine learning algorithm based on hand-crafted features and Gradient Boosted Trees (GBT). Using GBTs allows us to attain insights on feature importance, explain the classification results, and determine shortcomings of MSA simulations. Our second approach uses Convolutional Neural Networks (CNN). In contrast to GBT, CNNs only require minimal data processing, as they can automatically learn relevant features through training. However, to interpret these features, additional analysis is required. In the following, we introduce both machine-learning approaches to classification, and describe our training setups.

4.2.3 Training Classifiers

To assess the realism of simulations, we trained a GBT and a CNN classifier for each simulated data collection separately and independently. Each classifier takes as input an MSA or MSA features and outputs the label "simulated" or "empirical".

We trained each classifier using 10-fold cross validation and averaged over the respective 10 performance metrics. We used the *Balanced Accuracy* metric (BACC; see Section 2.3.3) to assess performance, as this metric allows for varying proportions of simulated/empirical MSAs in the data collection and better reflects classification accuracy for imbalanced datasets. The best BACC value is 1 and the worst value is 0. See Section 2.3.2 for further details on training machine learning models.

4.2.3.1 Gradient Boosted Trees

Gradient Boosted Trees (GBT) is an ensemble machine learning technique that combines multiple Decision Trees to obtain an accurate prediction model [59]. Training a GBT classifier consists of B sequential stages, with each stage contributing an additional Decision Tree that improves upon the estimator of the previous stage. For further details on GBTs, see Section 2.3.4.2.

Prediction Features

To classify MSAs into simulated or empirical ones, we computed 23 features for each MSA. Four of these features are MSA attributes: the *sites-over-taxa ratio*, the *patterns-over-taxa ratio*, the *patterns-over-sites ratio* and the proportion of invariant sites (*% invariant*). For data collections simulating indel events, we also used the proportion of gaps as a feature (*% gaps*). Further, we quantified the signal in the MSA using the difficulty of the respective phylogenetic analysis as predicted by Pythia [66] (*difficulty*; see Chapter 3), as well as the Shannon entropy [156] of the MSA (*Entropy*), a multinomial test statistic of the MSA (*Bollback Multinomial*; [20]), and an entropy-like metric based on the number and frequency of patterns in the MSA (*Pattern Entropy*). For further details on the computation of these metrics, see Chapter 3. We inferred 100 trees based on the fast-to-compute maximum parsimony (MP) criterion [44, 52] using RAxML-NG [96] and also included two features based on properties of the inferred 100 MP trees: the average pairwise topological distance using the Robinson-Foulds distance metric (RF_{MP}) [145], as well as the proportion of unique topologies (N^*_{MP}). We further refer to these features as *difficulty features*.

To assess downstream effects on tree inferences using simulated and empirical data, we inferred a single Maximum Likelihood (ML) tree using RAxML-NG [96]. For each MSA, we performed a single RAxML-NG tree inference based on a random starting tree using RAxML-NG's `--search1` execution mode. On the resulting ML tree, we executed the RAxML-NG `--eval` mode. In this execution mode, RAxML-NG does not alter the tree topology, but only optimizes the branch lengths and substitution model parameters. Based on the resulting ML tree, we computed a set of branch length features, namely the minimum, maximum, average, standard deviation, median, and sum over all branch lengths in the ML tree ($brlen_{min}$, $brlen_{max}$, $brlen_{avg}$, $brlen_{std}$, $brlen_{med}$, $brlen_{sum}$).

We used the next six features to highlight one of the recurrent problems of simulated sequence generators: a common simplification used in simulations is the assumption that substitutions occur at uniformly distributed random locations along the sequence, which appears not to be the case for real-world genetic data [25]. Thus, we expected empirical MSAs to be less uniform than simulated MSAs, and we henceforth attempted to confirm this hypothesis.

To quantify substitution frequency distributions along an MSA, we first inferred a single MP tree using RAxML-NG. Then, based on the parsimony criterion, we calculated the number of substitutions per site, resulting in a vector $\boldsymbol{m}$.

Given the corresponding vectors $\boldsymbol{m}$ for empirical and simulated MSAs, we can anecdotally observe that the locations of substitutions appear to be less uniformly distributed in empirical than in simulated MSAs (see Figure 4.2). To the best of our knowledge, there is no panacea in quantifying the absence of structure in data, and it is part of ongoing research in the field of cryptography. We resorted to the Fourmilab Random Sequence Tester (FRST) (`https://www.fourmilab.ch/random/`), that is used to evaluate pseudo-random number generators, to quantify randomness in $\boldsymbol{m}$. FRST computes six measures of randomness: Entropy ($Entropy_{rand}$), maximum

compression size reduction in percent (*comp*), Chi-Square test (Chi^2), arithmetic mean ($mean_{rand}$), Monte Carlo Value for Pi (*mcpi*), and Serial Correlation Coefficient (*SCC*) [93]. We executed FRST with a binary representation of $\boldsymbol{m}$ on all data collections. Then, we normalized the computed measures of randomness, and used these values in our predictions. We henceforth refer to this set of six features as *randomness features*.

Training and Optimization

For each of the simulated data collections presented above, we trained a distinct binary GBT classifier. Note that we trained distinct classifiers to ensure an unbiased estimate of the simulation realism per data collection separately. We trained each GBT classifier using a stratified 10-fold cross validation procedure. Here, stratified means that the proportion of empirical and simulated MSAs in both training and test subsets was the same. The training data consisted of one simulated data collection and the empirical data collection for the respective data type. We used the hyperparameter optimization framework Optuna [4] to determine the optimal set of hyperparameters for each classifier. For each GBT classifier, we performed 100 Optuna iterations using a Tree-structured Parzen Estimator algorithm [16] to sample the hyperparameter space.

To prevent the classifiers from overfitting the data, based on preliminary experiments, we limited the depth of the individual Decision Trees to a maximum of 10, the maximum number of leaves to 20, and the minimal number of samples per leaf to 30. Additionally, we applied L1 and L2 regularization to prevent overfitting and better generalize to unseen data [61] (see Section 2.3.2). We determined the optimal weights of L1 and L2 regularization independently using Optuna.

We generated the training data and features with a custom pipeline that we implemented using Snakemake [94] and Python 3. In our setup, we used RAxML-NG version 1.1.0 for the ML tree inference and the MP tree inferences, and PyPythia version 1.0.0 to compute the difficulty features. The pipeline code, instructions to reproduce the results, as well as the Apache parquet files containing all training data, are available in our GitHub repository at `https://github.com/tschuelia/SimulationStudy`. We trained all LightGBM GBT classifiers using a custom training script that is also available at the above GitHub repository, alongside all training results presented in our work.

4.2.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a popular prediction method originally developed for computer vision and image processing. Recently, they have been applied to predict properties of biological sequences [5, 9, 198]. A CNN jointly learns a representation of the data (through convolution layer(s)) and the classification of the data based on these representations (in our case using a fully connected layer). See Section 2.3.4.3 for a more detailed explanation of CNNs. Note that, in analogy to the GBT classifiers, we trained a distinct CNN per simulated data collection.

In the following, we provide a brief overview of the network and its architecture. All code required to reproduce the CNN training and classification results is available on GitHub at `https://github.com/JohannaTrost/seqsharp`.

MSA Representation

To obtain a numeric representation of an MSA, where the CNN is invariant to the order of sequences, we used a two-step approach. First we decided to represent the MSA using its site-wise AA or nucleotide composition, that is, the AA or nucleotide proportions per site, which sum to one. Second, each AA/nucleotide, as well as gaps are passed to the convolution network as input features (channels), resulting in 5 (4 DNA sites + gap) or 21 (20 AAs + gap) channels. This is analogous to using color channels in an image. It maintains the identity of a nucleotide/AA and is common practice when applying CNNs to biological sequences [9]. The input size was the maximum MSA length in the simulated and empirical data collection. All MSAs with fewer sites were zero-padded at their edges to match the fixed input size.

Empirical AA sequences typically start with Methionine (`M`), which simulations do not account for. Therefore, we removed the first and second sites from the empirical AA data to avoid biasing the prediction.

CNN Architecture

We developed two architectures, one for each data type (DNA and AA). We explored alternative architectures and chose the architecture with the best balance between complexity and performance. For AA MSAs, we used a single one-dimensional convolution layer with 210 filters of size 1×21 (kernel size $\times$ input channels). Of note, these filters do not consider the phylogenetic structure of the data, and simply capture AA profiles at single sites, instead of larger motifs spanning several contiguous sites that are typically used in CNNs. For DNA sequences, we used a two-layer CNN, whose first layer has 100 filters of size 3×5 and intends to capture codon structure. The second layer has 210 filters of size 1×100. A standard Rectified Linear Unit (ReLU) activation function is employed in both architectures. For both, DNA, and AA architectures, the layers following convolution comprise a dropout layer, which deactivates a node with a certain probability (here we chose 0.2) to avoid overfitting and global average pooling along the sequences. A final fully connected layer combines all features (channels) to generate the binary prediction. For this, we used a Sigmoid activation function. In total, the AA network counts 4831 learnable parameters, while the DNA network has 23 021 due to the additional convolution layer.

4.2.3.3 Performance Evaluation

Using the BACC metric per data collection, we compared the performance of pairs of classifiers of simulated data collections. To evaluate whether the difference of the BACCs of two data collections and therefore two different evolutionary models is significant, we conducted multiple unpaired two-samples t-tests, where one sample

consists of the validation BACC for each fold. This allowed us to compare models in their ability to simulate data that are more or less or equally realistic. For AA data, we compared the BACCs of the following groups: Poisson vs. WAG, WAG vs. LG, LG vs. LG+C60, and all pair-wise combinations of site heterogeneous models. The null hypothesis is that these models yield equal average BACCs across folds. We rejected the null hypothesis if the resulting P-value was below the significance level of 0.05. For DNA data, we compared the BACCs of JC vs. HKY, HKY vs. GTR, GTR vs. GTR+G and GTR+G vs. GTR+G+I. To account for multiple testing, we applied a Bonferroni correction [2]. This means that we multiplied each P-value by the number of tests for each data type separately.

4.3 Results

4.3.1 Classification Accuracy

Table 4.1 shows the BACC for our GBT and CNN classifiers across all data collections. Both classifiers were able to accurately distinguish simulated from empirical data. The GBT classifiers achieved high BACCs for all simulated AA data collections (≥ 0.98), as well as for all gapless DNA data collections (≥ 0.89). We observed the worst BACC of 0.77 for the DNA data collection simulated under GTR+G+I with gaps simulated according to the *mimick* procedure. The CNN classifiers achieved BACCs ranging from 0.93 to 0.9996. Interestingly, the GBT classifiers showed similar BACCs or even outperformed the CNN on the AA data collections, but achieved lower BACCs on DNA collections.

On DNA data collections, substitution models with fewer degrees of freedom than the GTR model (JC and HKY) were classified more accurately (BACC = 0.99 for CNN and BACC = 0.96 for GBT). However, increases in model complexity did not always translate into improvements in data realism. For instance, the performance of the CNN was marginally better on simulations under the HKY model than on simulations under the simpler JC model (P-value = 0.03). The GBT predictions, which were equally accurate for JC and HKY simulations (BACC = 0.96), did not reflect any improvement in the simulations due to more degrees of freedom in the HKY model either. Moreover, the CNN yielded the lowest BACC (0.93) on simulations conducted under the GTR model. In contrast, simulations that included rate heterogeneity (GTR+G) were slightly easier to classify (BACC = 0.94, P-value = 0.04). Contrary to our expectations, including a proportion of invariant sites (GTR+G+I) did not result in a lower BACC compared to GTR+G simulations (BACC = 0.94, P-value = 1.0 for CNN, BACC = 0.89, P-value = 1.0 for GBT).

We did not observe the expected trend of improved realism with increasing model complexity for the AA data collections. For instance, the CNN showed the lowest BACC on simulations under the LG substitution model (BACC = 0.95) and not on the more complex mixture models. For the GBT, distinguishing the LG+S256+G4 data collection appeared to be easier than the data collection based on the simpler LG+C60 model (P-value = 0.77). Unexpectedly, all simulations using a mixture of

Data collection	BACC	
	GBT	CNN
DNA data collections		
JC	0.96	0.99
HKY	0.96	0.99
GTR	0.94	0.93
GTR+G	0.89	0.94
GTR+G+I	0.89	0.94
GTR+G+I+mimick	0.77	0.97
GTR+G+I+sparta	0.94	0.97
AA data collections		
Poisson	0.99	0.9996
WAG	0.99	0.97
LG	0.99	0.95
LG+C60	0.98	0.99
LG+S256	0.99	0.995
LG+S256+G4	0.99	0.99
LG+S256+GC	0.98	0.99
LG+S256+GC+sparta	0.99	0.996

Table 4.1: Average of the BACC on empirical and simulated data collections across 10 folds for the GBT and CNN classifiers. Parameter configurations of simulations listed in the first column are sorted with increasing complexity from top to bottom for both DNA and AA data. The last row(s) per section (DNA and AA) shows results on data collections with indels.

stationary frequency profiles (LG+C60, LG+S256, LG+S256+G4 and LG+S256+GC) were nearly perfectly discriminated from the empirical data collection with both GBT and CNN (BACC ≥ 0.98). With the CNN, we did not observe a significant performance difference between these evolutionary models (P-value ≥ 0.38).

To rule out the possibility that these rather unexpected findings are a consequence of specific behaviors inherent to the AliSim simulator, we conducted an experiment to evaluate the performance of the CNN classifier pre-trained with LG+S256 simulations on data generated using a simulator developed in the LBBE research group that employs the same model. Our results indicated that the CNN classifier performed comparably well on the alternative simulations (BACC = 0.99). In addition, we tested the same CNN on simulations using 4096 profiles. These simulations were only slightly harder to classify (BACC = 0.98) than the ones based on only 256 profiles (BACC = 0.995).

The CNN trained on empirical data collections with indels and simulations under the most complex evolutionary model with indels (LG+S256+GC+sparta, GTR+G+I+mimick, GTR+G+I+sparta) also yielded highly accurate predictions (BACC = 0.996 for AA and BACC > 0.97 for DNA data). The results were similar to or better than the results obtained without indels. There was no significant difference between CNN

Data collection	BACC	Three most important features
DNA data collections		
JC	0.96	SCC, $\text{brlen}_{\text{stdev}}$, Entropy
HKY	0.96	SCC, $\text{brlen}_{\text{stdev}}$, Entropy
GTR	0.94	SCC, $\text{brlen}_{\text{stdev}}$, Entropy
GTR+G	0.89	SCC, Pattern Entropy, $\text{brlen}_{\text{sum}}$
GTR+G+I	0.89	SCC, Pattern Entropy, $\text{brlen}_{\text{sum}}$
GTR+G+I+mimick	0.77	SCC, % invariant, patterns-over-sites ratio
GTR+G+I+sparta	0.94	SCC, % gaps, $\text{mean}_{\text{rand}}$
AA data collections		
Poisson	0.99	SCC, $\text{brlen}_{\text{med}}$, Bollback Multinomial
WAG	0.99	SCC, $\text{brlen}_{\text{med}}$, % invariant
LG	0.99	SCC, $\text{brlen}_{\text{med}}$, % invariant
LG+C60	0.98	SCC, Entropy, patterns-over-sites ratio
LG+S256	0.99	SCC, Entropy, % invariant
LG+S256+G4	0.99	SCC, Entropy, patterns-over-sites ratio
LG+S256+GC	0.98	SCC, Entropy, patterns-over-sites ratio
LG+S256+GC+sparta	0.99	patterns-over-sites ratio, SCC, Entropy

Table 4.2: Average BACC on empirical and simulated data collections across 10 folds. Parameter configurations of simulations listed in the first column are sorted by increasing model complexity from top to bottom for both, DNA, and AA data. The third column lists the three most important features for classifying the data.

performance on the two DNA indel models employed (P-value = 1.0). Simulating indels increased the GBT classification accuracy for AA data (BACC = 0.99) and the *sparta* based DNA data collection (GTR+G+I+sparta; BACC = 0.94) compared to the same model of evolution without indel simulations (LG+S256+GC BACC = 0.98; GTR+G+I BACC = 0.89). We did, however, observe a significant decrease in accuracy comparing the two DNA indel models (P-value $\ll 10^{-300}$). The GBT classified the GTR+G+I+sparta data collection with high accuracy (BACC = 0.94), but showed an unexpectedly low BACC of 0.77 for GTR+G+I+mimick.

4.3.2 Feature Importance

4.3.2.1 GBT

To gain insights into why the classification task achieved high prediction accuracy and appears to be rather easy in general, we assessed the influence of the described features on the prediction of the GBT classifiers. To this end, we computed the gain-based feature importance. The gain-based feature importance directly measures the contribution of a feature to the reduction of the loss function (see Section 2.3.5). Table 4.2 shows the prediction accuracy as well as the three most important features for the gradient boosted tree classifiers. The features are sorted by importance, that is, the most important feature is listed first.

We observed that, except for one data collection, the SCC randomness metric was the most important feature. For classifying the LG+S256+GC+sparta data collection, it was the second most important feature. Figure 4.3 shows the distribution of SCC values for one example DNA data collection (GTR+G+I), as well as for one example AA data collection (LG+S256+GC) compared to the distribution for the respective empirical data collections. The lower the SCC value, the more random the distribution of rates of evolution across sites in the MSA is. The SCC values for simulated MSAs are substantially lower than for empirical MSAs. This indicates that the rates of evolution across sites are more uniformly distributed in simulated MSAs compared to empirical MSAs, simulated data is thus more "random" than empirical data. We also observed similar patterns for all other data collections.

We also frequently observed the *Entropy*, the *Pattern Entropy*, as well as the *Bollback Multinomial* metrics as being among the three most important features. While the randomness features measure the randomness across *sites* of the MSA, these three features quantify the randomness across *taxa* per site, indicating that simulated data are not only more "random" across sites, but also within sites.

To gain further insights into the importance of the randomness features for classification, we retrained all GBT classifiers without this set of randomness features. As expected, the BACCs decrease for all data collections. Interestingly, the BACCs for the GTR+G and GTR+G+I DNA data collections decreased substantially from 0.89 to 0.65 and 0.61 respectively, yielding a prediction that is only marginally better than random guessing. Using this reduced set of features for the prediction, we observed interesting differences in feature distributions. We observed that, compared to simulated data, empirical data tend to exhibit a higher proportion of invariant sites (Figure 4.4(a)). The branch lengths in trees inferred for simulated MSAs tend to be shorter (Figure 4.4(b); for better visualization, we only show data below the 90% percentile), and the RF_{MP} tend to be higher for empirical data (Figure 4.4(c)). While Figure 4.4 depicts the distribution of feature values for one exemplary data collection (JC) only, these observations apply to all simulated data collections. The more complex the model of evolution, the less pronounced these differences become, especially for the simulated DNA data under GTR+G and GTR+G+I. It is noteworthy however that even GTR+G+I, which contains a dedicated parameter for modelling the proportion of invariable sites, yields MSAs with fewer invariant sites than in empirical data.

Compared to the remaining data collections, we observed a substantially worse BACC of 0.77 for the GBT of the GTR+G+I+mimick data. To investigate this phenomenon, we split MSAs site-wise into 100 parts (buckets), averaged the number of substitutions per bucket (normalized by the maximum number of substitutions per MSA), and averaged the buckets over every MSA. Interestingly, we could observe that the substitutions for empirical and the GTR+G+I+mimick data collections are concentrated at the beginning and the end of the MSAs, while the number of substitutions in GTR+G+I+sparta appear to be uniformly distributed. This also appeared to be the case for other substitution models according to our analyses.

This result is in agreement with Bricout *et al.* [25], who also found this pattern in a large-scale analysis of empirical MSAs.

As described above, we simulated the DNA data collections and the AA data collections without indels based on trees inferred using RAxML-NG. Trees for AA data with indels used for our indel simulations were inferred using IQ-TREE. For 10 out of 15 data collections, one of the branch length features was among the three most important features. To ensure that we did not capture an inference tool induced bias in our prediction, we retrained all classifiers using only the MSA-based features by discarding all branch length features. We observed no substantial impact on overall prediction accuracies. With GTR+G+I+mimick we observed the highest BACC difference. Using all features, the GBT achieved a prediction accuracy of 0.77. Discarding the branch length features resulted in a BACC of 0.74.

4.3.2.2 CNN

In addition to the feature analysis of the GBTs, we further investigated the remarkably accurate performance of the CNN on simulations using mixtures of stationary frequency profiles (the S256 or C60 model). Given that we could achieve better performance when using average global pooling, that is, averaging across the sequence, instead of maximum local pooling following the convolution layer (see the paragraph on CNN Architecture), we hypothesized that there must be predictive global features that aid in distinguishing simulated from empirical MSAs. In particular, we hypothesized that MSA-wise frequencies of AAs or nucleotides may differ between simulated and empirical data. To test this hypothesis, we trained logistic regression models to undertake the same classification task, but using site compositions averaged along the MSA, that is, MSA compositions. Figure 4.5 shows that the logistic regression model indeed performed well, particularly for simulated data under mixture models (BACC > 0.94). Moreover, across collections, there is a strong correlation between BACCs of the CNNs and the logistic regression models ($r^2 = 0.85$). We also attempted to train the logistic regression model on DNA data simulated under the GTR+G+I model, but found that there was no significant improvement during the first 100 epochs (BACC $= 0.51$). Therefore, the MSA composition is not informative for the classification of DNA data, but highly informative for AA data.

4.3.3 Classification Accuracy and Pythia Difficulty

For both, GBT, and CNN classifiers, we observed a general trend for lower classification accuracy on more difficult MSAs according to the Pythia difficulty score. The higher the Pythia difficulty for an MSA, the weaker the signal in the data and the more difficult it is to obtain a well-supported phylogeny, as the likelihood surface exhibits multiple likelihood peaks that are indistinguishable by standard phylogenetic significance tests [66]. In addition to assessing the BACC as a function of the difficulty of *simulated* MSAs, we also assessed the BACC as a function of the difficulty of the underlying *empirical* MSAs. For MSAs with a higher Pythia difficulty, it should be more difficult to find the true phylogeny, as the likelihood surface exhibits multiple peaks. However, simulating an MSA requires a reference phylogeny and

relying on a "bad" tree might have a negative impact on the realism of the simulated data. If this holds true, the classification of simulated MSAs based on easy empirical MSAs (that is, simulations based on "good" trees) could be more difficult, leading to a lower BACC than the classification of simulated MSAs that are based on difficult empirical MSAs. Interestingly, we observed the opposite effect: the more difficult the underlying empirical MSAs, the lower the BACC.

Figure 4.6 depicts this observation for the simulated data collections with the lowest BACC for GBT (GTR+G+I+mimick) and CNN (LG) respectively. Both Figures show the BACC as a function of the Pythia difficulty over the *simulated* MSAs (left panels), as well as the BACC as a function of the Pythia difficulty over the underlying *empirical* MSAs (right panels). The colors indicate the number of MSAs per difficulty range on a log-scale. All four examples demonstrate a substantial decrease in BACC with increasing difficulty. Note that the LG data collection only contains 9 simulated MSAs with a Pythia difficulty ≥ 0.6. The CNN misclassifies 5 of these MSAs as being empirical, resulting in the decrease of the BACC in the right tail of the left subplot of Figure 4.6(b). A similar effect causes the decrease of the BACC in the right tail of the right subplot of Figure 4.6(b): only 12 empirical MSAs have a Pythia difficulty ≥ 0.7, out of which the CNN misclassifies 6 MSAs. Taking this into account, the decrease of BACC with increasing difficulty is overall more pronounced for the GBT on the DNA data collection.

We suspect that the decrease in BACC with increasing difficulty is related to the amount of information in the data: MSAs with low information do not only yield inconclusive phylogenetic analyses (as indicated by the high Pythia difficulty), but also lack a strong signal indicating their realism. For instance, an MSA for a highly conserved gene essentially only contains the information about a single sequence because all sequences are nearly identical, which yields phylogenetic reconstruction and classification challenging. An MSA for a less conserved gene contains more information, which can be leveraged for both, phylogenetic reconstruction, and classification. In the extreme case, an MSA where all sites are constant would obviously be difficult to use for both tasks.

To test this hypothesis, we computed a sequence similarity measure for each MSA in the GTR+G+I+mimick DNA data collection, and the LG AA data collection. To this end, we computed the sequence similarity of an MSA as the median pairwise Hamming distance of all pairs of sequences. We normalized the Hamming distance to a zero-one-scale by dividing by the sequence length. The lower this median hamming distance, the more similar the sequences in the MSA are. We then computed the Pearson correlation coefficients between the sequence similarities and the difficulties of the simulated MSAs, and between the sequence similarities and the difficulties of the empirical MSAs. We performed these analyses for both data collections individually. For the GTR+G+I+mimick DNA data collection, we observe Pearson correlation coefficients of -0.39 and -0.33 for simulated and empirical MSAs respectively (P-value $< 10^{-20}$ in both cases). This confirms our hypothesis that the difficulty tends to increase with increasing sequence similarity. However, for the LG AA data collection, we could not confirm our hypothesis: the Pearson correlation

is only -0.07 (P-value $< 10^{-7}$) for the sequence similarities and the difficulties of the simulated MSAs, and the Pearson correlation is insignificant in correlating the difficulties of the underlying empirical MSAs to the sequence similarities (correlation coefficient 0.01 with a P-value of 0.65). However, as stated above, the LG data collection, as well as all remaining simulated gapless AA data collections, comprise only but a few difficult MSAs (Pythia difficulty ≥ 0.7), leaving little opportunity to unravel a significant correlation.

4.4 Discussion

In this study, we assessed the realism of sequence simulations by attempting to discriminate between simulated MSAs and empirical MSAs using two distinct and independently developed classification methods. Specifically, we evaluated and interpreted the predictive accuracy of these approaches as a measure of realism. By addressing this question, we aimed to gain insights into the ability of current evolutionary models to accurately simulate evolutionary processes using Continuous Time Markov Chains (CTMC). The ability to accurately model sequence evolution and thus simulate realistic MSAs is crucial both for the evaluation of inference tools and the development of neural density estimation techniques for inference.

Note that producing MSAs that are indistinguishable from empirical ones is a necessary, but not sufficient, condition for the degree of realism of the underlying model. First, poor classification performance can occur because the classifier deploys inappropriate functions or data representations. Hence, one cannot guarantee that the simulated MSAs are realistic under all possible criteria. Second, poor performance can also be induced by optimization issues, especially when using deep learning methods. During our experiments, we observed low accuracies for CNNs several times. We managed to alleviate this by adapting the learning rate, the number of filters, or the pooling method, for instance. We thus advise researchers interested in classification performance as a realism metric to closely monitor indicators of poor optimization, in particular, learning curves and gradient norms – in our case, poor optimization also induced a larger variance across folds and accuracy discrepancies for the two classes. Because we found that all simulated MSAs were easy to discriminate from empirical MSAs, and because our results are consistent across two technically substantially distinct and independent classification methods, we conclude with confidence that the simulated MSAs generated in our study are not realistic.

It is worth noting here that we originally chose to develop a CNN for the classification task, as it can capture local dependencies among sites. With a kernel size greater than one, the network could potentially benefit from these dependencies for classification, as they are present in empirical MSAs yet cannot be replicated with standard site-independent models of sequence evolution. However, we discovered that for AA data, the CNN yields accurate performance, even with a kernel size of one in combination with global average pooling (as an alternative to the commonly used local maximum pooling). This type of network primarily focuses on capturing global features while overlooking local among-site dependencies. Consequently,

these choices enabled us to thoroughly explore the limitations of current sequence simulations and different evolutionary models beyond their unrealistic assumption of independently evolving sites. However, in the future, a CNN architecture could be deployed to assess the importance of local site dependencies that are not accounted for in current state-of-the-art simulators.

Our study uses two fundamentally different classifiers, which allows for a broader assessment of possible weaknesses of current sequence evolution simulations: GBTs rely upon diverse, yet well-defined MSA properties, such as branch lengths or the randomness features that capture the assumption of homogeneity along MSAs in standard simulations. Given the high feature importance of the evolutionary rates (SCC) in the MSA, our GBTs exploit a lack of structure along simulated MSAs. The CNN only considers site-wise composition vectors, and thus exploits a signal that is not directly exploited by the GBTs. Furthermore, for the classification we used diverse and representative empirical AA and DNA databases: TreeBASE comprises representative data sets that are commonly analyzed in the field because it only contains MSAs of published studies, whereas HOGENOM offers a diverse sample of existing data, drawing from 499 nuclear Bacterial genomes, 46 from Archaea, and 121 from Eukaryotes.

The structure detected by our GBTs in empirical nucleotide MSAs from TreeBASE is not due to the type of genetic code present. We computed the number of stop codons in all genes in the database and at all three phases, and did not observe an excess of MSAs with 0 or 1 stop codons per sequence (Figure 4.7). However, in the future it will be interesting to investigate the realism of existing codon models, on a data set of coding DNA sequences.

We used phylogenetic trees reconstructed from these empirical data collections to simulate data as realistically as possible. Thereby, we circumvented having to simulate realistic trees and can invoke simulations that are as similar as possible to the empirical MSAs. However, it is important to note that the realism of the simulations depends on the quality of the inferred phylogenetic trees when deploying this procedure. Since we do not know the true trees for the empirical MSAs, we must acknowledge that there is some uncertainty or error in the inferred trees that the simulations inherit. Hence, at least part of the classifier accuracy, that is, part of the difference between the simulations and the empirical MSAs, could be attributed to the difference between the inferred trees and the unknown true trees. However, our choice to use ML trees inferred under the same models used for the subsequent simulation (except for the AA data, see below) may constitute the most realistic approach toward generating MSAs that resemble empirical MSAs. Indeed, the best-known ML tree $\hat{T}$ under the model $\boldsymbol{M}$ for an MSA $\boldsymbol{S}$ is the best tree we can find that maximizes the probability of observing $\boldsymbol{S}$. Any other tree is less likely to have generated $\boldsymbol{S}$ under the model $\boldsymbol{M}$ (assuming that the optimization did find the ML tree). Therefore, by simulating under model $\boldsymbol{M}$ along the tree $\hat{T}$, we maximize the probability (or get close to maximizing it) of generating MSA $\boldsymbol{S}$. We expect that thereby, we also obtain a high probability of generating MSAs that resemble $\boldsymbol{S}$, that is, MSAs that "look" empirical.

However, for AA data, the inference of trees from AA MSAs without indels was performed under the LG substitution model. The resulting trees may be different from the ML tree obtained under the WAG model or under mixture models. In particular, trees inferred under the LG model may have branches that are too short to be used for simulating MSAs with site-heterogeneous mixture models because inferences under mixture models typically yield longer branches than inferences under the LG model. However, looking at the per-site AA diversity (Figure 4.8) reveals that sites simulated under mixture models appear more similar to empirical sites than sites simulated under the LG model. Therefore, it remains unclear why mixture models failed to improve MSA realism according to our classifiers. Overall, for some of our experiments on AA data, the mismatch between substitution models used to infer the trees, and those employed to simulate the MSAs, may be consequential and warrants further investigation.

The classification task was not difficult, neither for DNA nor for AA data. Our CNN achieved an average BACC of 0.98 across all evolutionary models. This shows that existing models of sequence evolution fail to capture important characteristics of empirical site-wise compositions. In turn, this questions to which extent previous results obtained on simulated data apply to empirical data.

We originally hypothesized that with increasing evolutionary model complexity, classification performance would decrease. However, our results do not fully confirm this initial hypothesis. On the contrary, both classifiers remained highly accurate under the most complex evolutionary models for AA simulations with heterogeneous stationary distributions across sites. In DNA simulations, the inclusion of rate heterogeneity and a proportion of invariant sites did not induce a substantial decrease in CNN classification accuracy, either. Using the HKY substitution model instead of the JC model did also not result in more realistic simulations as a function of observed classification performance. Finally, the most simple models, JC and Poisson, were classified with ease.

Future studies may help characterize the influence of the trees used for simulating MSAs on their realism. For instance, experiments where we simulate data using complex models of sequence evolution, and using simpler models on the same trees, may help us to characterize the ability of our classifiers to distinguish between different models, when the phylogeny is not a confounding factor.

We used a state-of-the-art indel model with individual parameters for insertions and deletions and sampled indel parameters from approximated joint distributions. Nevertheless, both classifiers could again easily distinguish simulated from empirical MSAs. In fact, classification accuracy substantially increased on DNA data *with* indels compared to DNA data without indels (GTR+G+I). In contrast, using the *mimick* procedure to superimpose gaps onto simulated data appeared to result in more realistic MSAs. Yet, these MSAs could still be easily identified as simulated ones based on their site-wise compositions, as shown by the CNN results.

Furthermore, the prediction accuracy for AA data tended to be higher than the prediction accuracy for DNA data. We suspect that this is due to the higher number

of states in the AA alphabet and therefore the increased number of possible patterns in an AA MSA, which makes it harder to simulate realistic data.

Our findings suggest that existing evolutionary models might not be able to generate data collections that appropriately resemble global low-level site composition features of empirical DNA or AA data collections using standard site- and position-independent CTMC. Considering the high importance of randomness related features for the GBT classifiers, and the respective feature value distributions, we conclude that the rate of evolution across sites of simulated MSAs are generated more uniformly along the MSA compared to empirical MSAs. For instance, we found that current models cannot reproduce the serial correlation of evolutionary rates that is present in empirical MSAs. We further observe that the proportion of invariant sites in standard simulations reduces their realism as measured by GBT. In addition, the CNN results reveal that simulated MSAs have unrealistic properties in terms of site-wise compositions that are independent of correlations among neighboring sites.

The unexpectedly high accuracy of the logistic regression model on simulations under mixture models that produce heterogeneous stationary distributions across sites indicates that these models simulate MSAs with an average MSA composition which is distinct from that of empirical data. This is particularly surprising for the LG+S256 models, which had been trained on HOGENOM data [153]. This discrepancy is unlikely to arise from simulating on trees inferred under the LG model rather than mixture models. Indeed, shorter branches in the LG trees should result in lower per-site AA diversity. However, we did not observe this in our data collections, as sites in simulations under the LG model have slightly higher AA diversity than those in empirical data (Figure 4.8). Moreover, the site-wise AA diversity appeared similar between simulations under LG+S256 and empirical data. The causes for the discrepancy in average MSA compositions need to be further investigated.

We believe that in the years to come, learning-based, likelihood-free approaches are likely to be more widely used in our field. Especially, if their performance (both in terms of phylogenetic reconstruction accuracy and runtime) is superior. However, we further believe that likelihood-based inference will continue to play an important role in the area of computational phylogenetics, as the statistical properties of ML and Markov chain Monte Carlo (MCMC) methods for posterior estimation still benefit from a better empirical knowledge.

Looking forward, this work paves the way for approaches to simulate more realistic MSAs by developing more realistic models of sequence evolution. We conclude that a substantial amount of research remains to be conducted to improve substitution as well as indel evolution models, for both, AA, and DNA data.

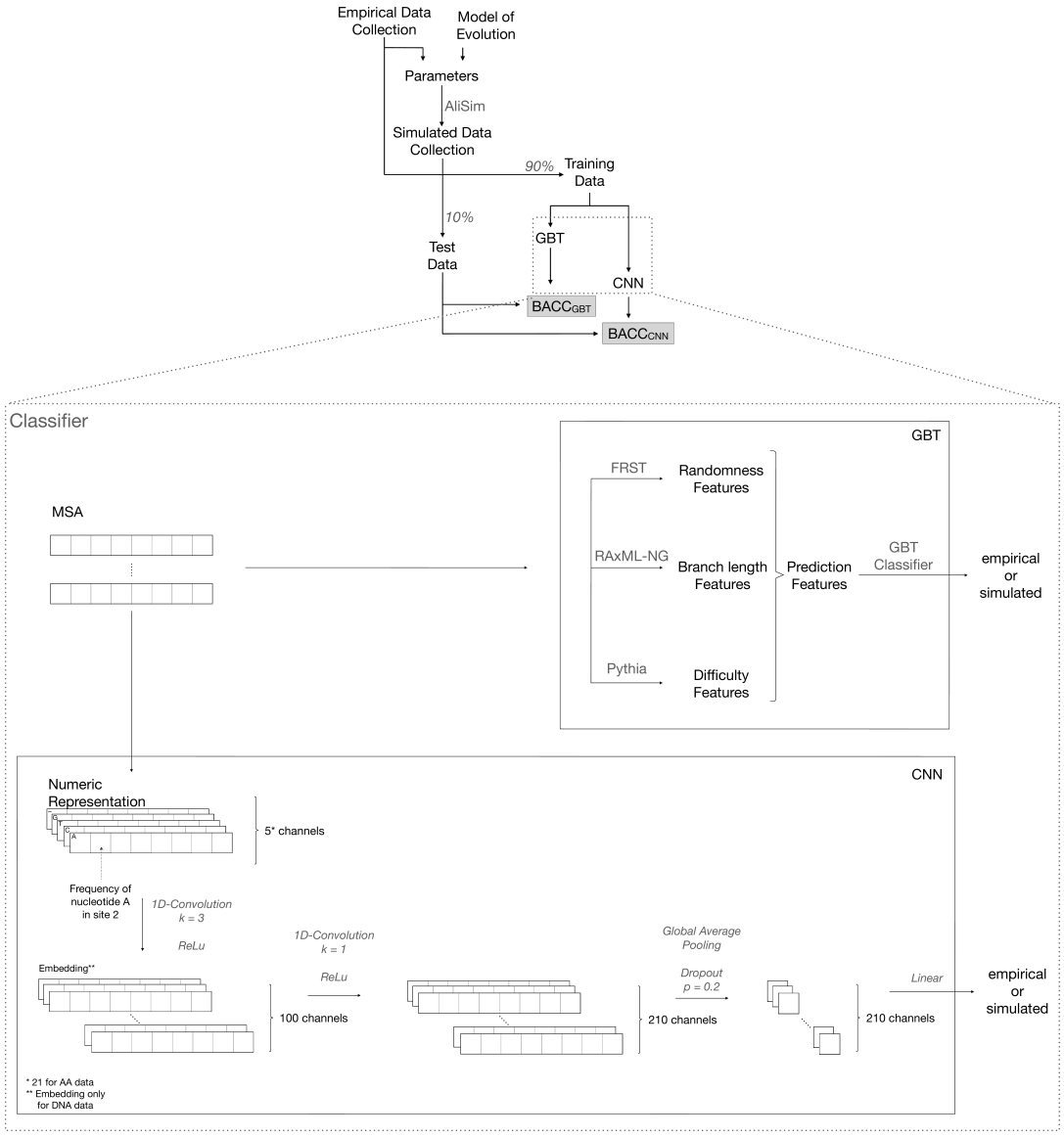


Figure 4.1: Schematic overview of our experimental setup. Based on a set of empirical MSAs (empirical data collection), we determined parameters for sequence simulation and simulated new MSAs (simulated data collection) under a given model of evolution using AliSim. Using the empirical and simulated data collections, we trained two distinct classifiers: a Gradient Boosted Tree (GBT) and a Convolutional Neural Network (CNN). The goal of both classifiers is to distinguish empirical from simulated MSAs. For training and evaluating our classifiers, we used a 10-fold cross validation procedure (not depicted for simplicity). In each fold, 90% of the data were used for training and 10% were used for performance evaluation. We evaluated the overall performance of the classifiers via the BACC.

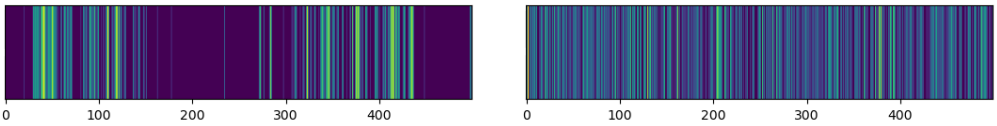
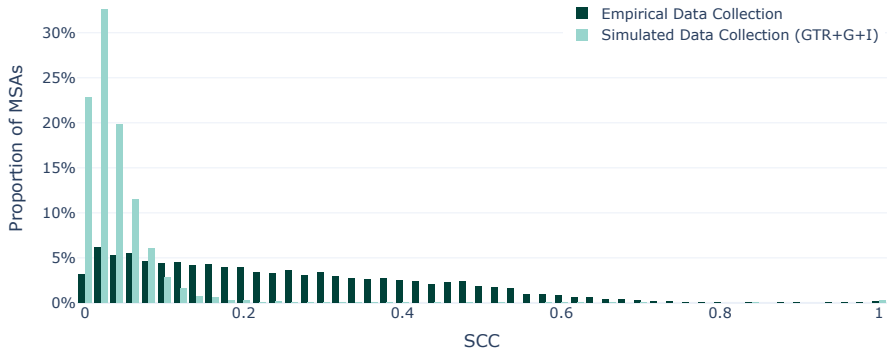
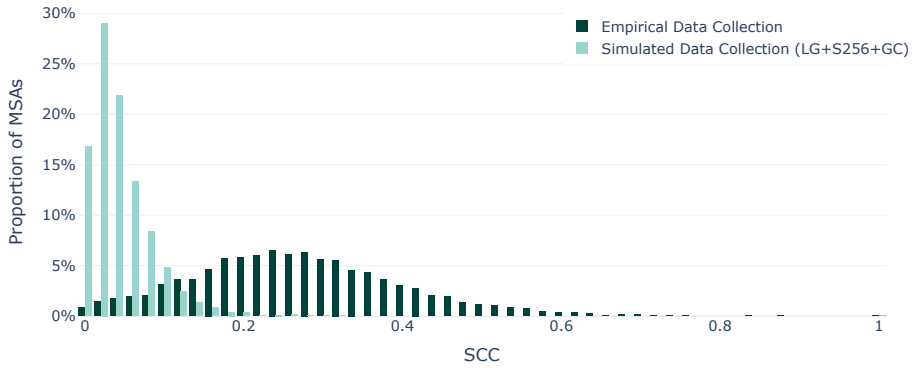


Figure 4.2: Visualized substitution rates for an anecdotal (specifically selected to highlight the issue) gapless empirical DNA MSA (left), and gapless simulated MSA (right) generated based on the inferred tree and estimated evolutionary model parameters of the left MSA under the GTR model. The x-axis denotes the MSA site index. A brighter color denotes more substitutions.

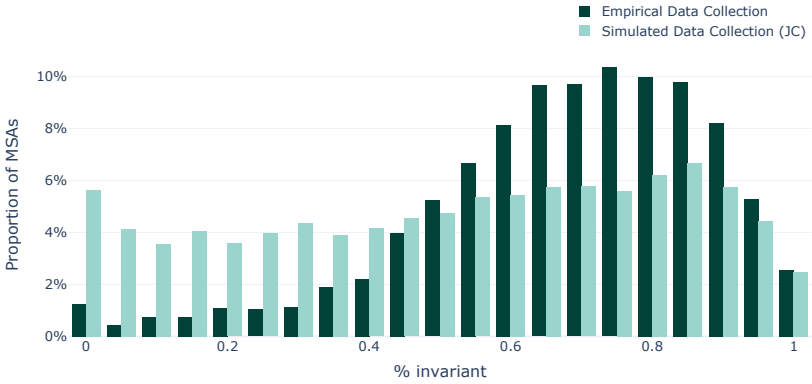


(a) Distribution of SCC values for the GTR+G+I and empirical DNA data collections.

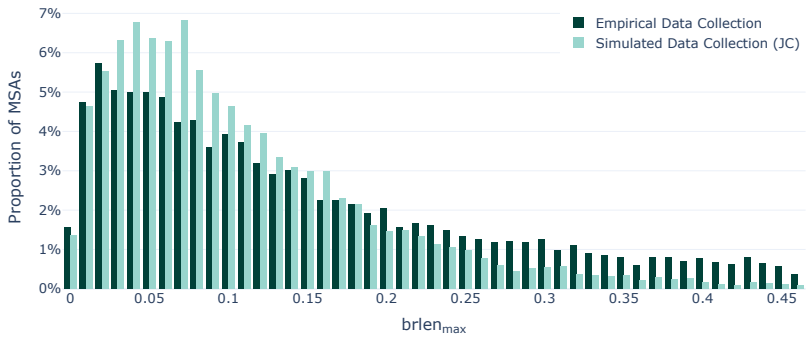


(b) Distribution of SCC feature values for the LG+S256+GC and empirical AA data collections.

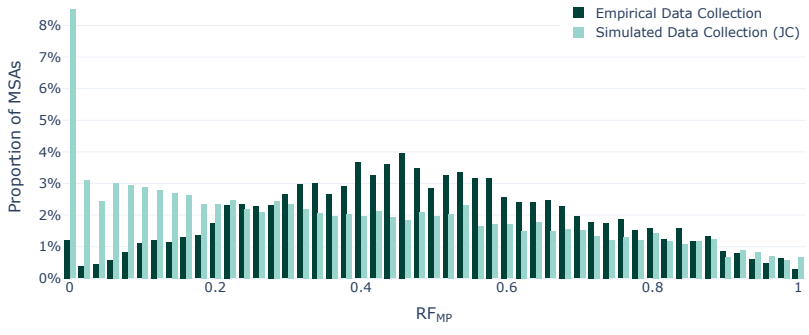
Figure 4.3: Feature distribution of SCC feature values for one exemplary DNA and AA data collection. The dark bars represent the respective empirical data collection and the light bars represent the respective simulated data collection.



(a) Distribution of proportion of invariant feature values for the JC and empirical data collections.



(b) Distribution of $brlen_{max}$ feature values for the JC and empirical data collections.



(c) Distribution of RF_{MP} feature values for the JC and empirical data collections.

Figure 4.4: Feature distribution for important features for classifying the JC data collection. The dark bars represent the empirical data collection and the light bars represent the simulated JC data collection.

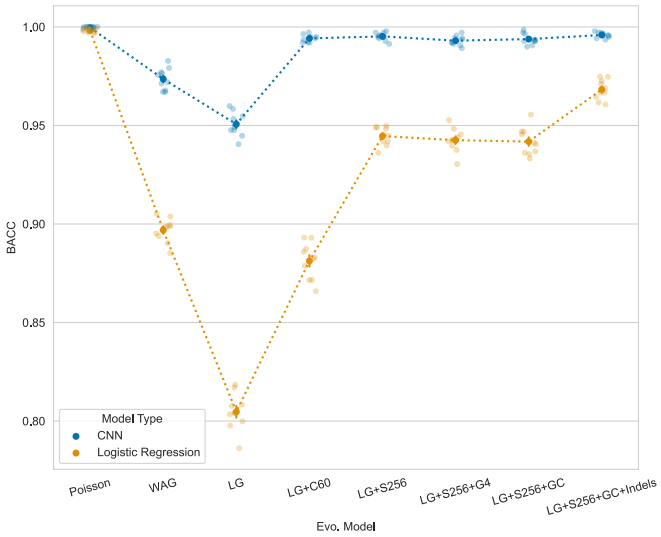
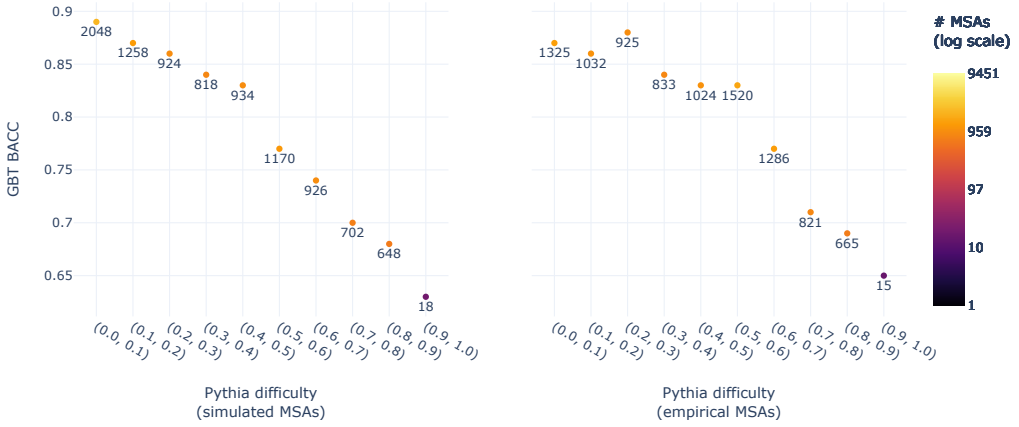
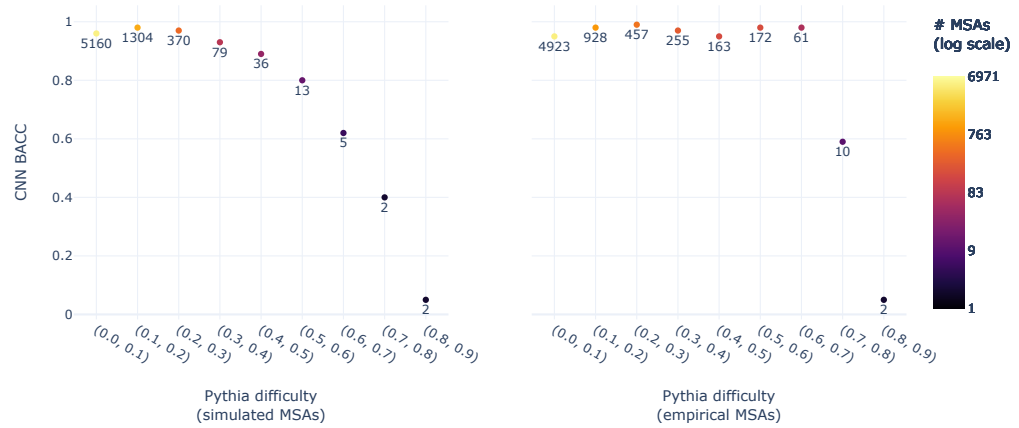


Figure 4.5: Performance of logistic regression on MSA compositions and CNN on site-wise compositions. For each evolutionary model, the BACC of each fold is represented, as well as the mean and standard error.



(a) BACC of the GBT classifier on the GTR+G+I+mimick data collection as a function of the Pythia difficulty of the *simulated* MSAs (left panel) and the underlying *empirical* MSAs (right panel).



(b) BACC of the CNN classifier on the LG data collection as a function of the Pythia difficulty of the *simulated* MSAs (left panel) and the underlying *empirical* MSAs (right panel).

Figure 4.6: The accuracy of the GBT and CNN classifiers as a function of the Pythia difficulty of the underlying MSAs. The number of MSAs per difficulty range is indicated by colors (log-scale), as well as text annotations.

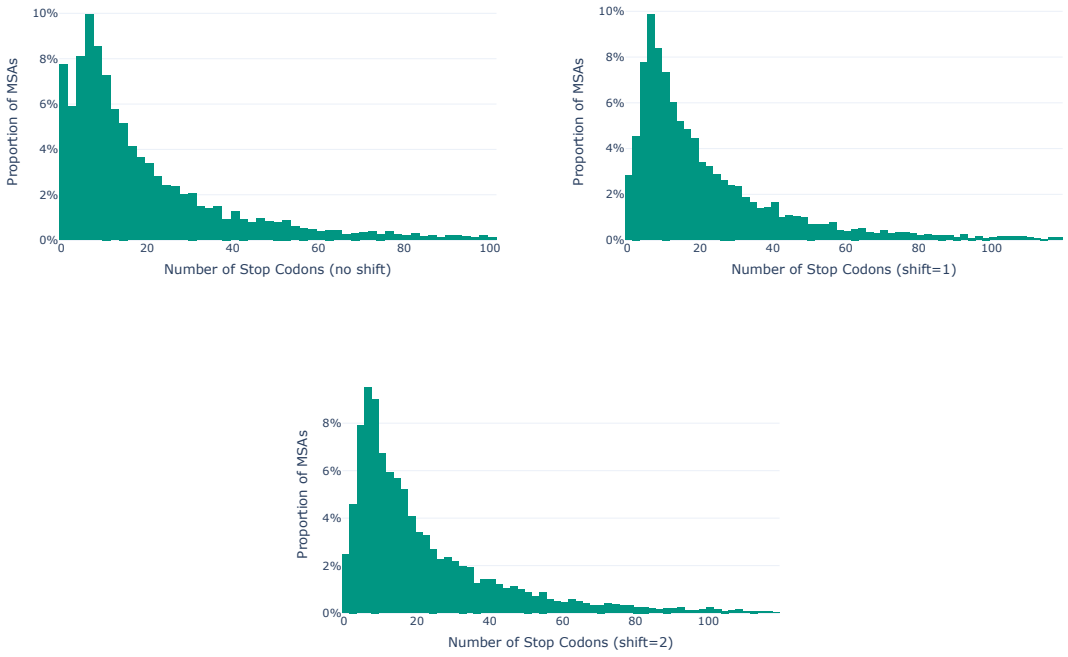


Figure 4.7: Number of stop codons in all translated DNA sequences of all MSAs of the empirical TreeBASE data collection. To ensure that we are not missing stop codons due to a shift in the reading frame, we computed the number of stop codons without shift, with one, as well as with two shifts in each sequence.

Figure 4.8: Site-wise AA diversity as number of unique AAs per site.

5. Pandora: Estimating Dimensionality Reduction Stability of Genotype Data

This chapter is derived from the peer-reviewed open-access publication:

> **Julia Haag**, Alexander I. Jordan and Alexandros Stamatakis. "Pandora: A Tool to Estimate Dimensionality Reduction Stability of Genotype Data." *Bioinformatics Advances*, Volume 5, Issue 1, March 2025. https://doi.org/10.1093/bioadv/vbaf040

Julia Haag designed, implemented, and evaluated the algorithm. All text and figures in this chapter were created by Julia Haag. Alexander I. Jordan provided statistical knowledge and helped design the *Pandora Support Values*. Alexandros Stamatakis contributed background knowledge, discussion, and feedback.

5.1 Background and Motivation

Dimensionality reduction techniques such as Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) are routinely deployed in numerous scientific fields as they facilitate data visualization and interpretation. Both methods project high-dimensional data to lower dimensions, while preserving as much variation, and thus information, as possible. Ever since the first application of PCA to genetic data almost half a century ago [118], PCA as well as alternative dimensionality reduction techniques have been widely used in modern population genomic analyses, including ancient DNA studies, to draw conclusions about population structure, genetic variation, or demographic history. For example, Hughey *et al.* [83] found that the Minoan

civilization did not originate from Africa, relying on PCA results. However, due to common challenges such as missing data and noise, the deployment of dimensionality reduction techniques to analyze ancient DNA or conduct contemporary population genetics studies is controversial. For instance, Yi and Latch [196] demonstrated that missing data can bias PCA analyses. Thus, quantifying the intrinsic uncertainty is pivotal to conclusive and cautious result interpretation. For example, the relative location of a population or an individual in a PCA/MDS embedding determines its relation to other populations or individuals. If there exists a substantial uncertainty regarding the projection location for some, or even all individuals in a study, the respective conclusions may potentially be erroneous and hence misleading. To the best of our knowledge, there currently exists no method to quantify the inherent uncertainty of dimensionality reduction techniques in either ancient or contemporary population genetics studies. To address this challenge, we introduce Pandora, an open-source tool that estimates the uncertainty of PCA and MDS analyses of population genetics and ancient DNA genotype datasets. Pandora estimates the uncertainty via bootstrapping over the Single Nucleotide Polymorphisms (SNPs). In addition to an overall, global stability score, Pandora also estimates the stability of a subsequent k-means clustering based upon the computed embeddings. Both stability scores have values that range between zero and one. Higher values indicate a higher stability and thus a lower uncertainty regarding the computed embedding for the respective dataset. Beyond this, Pandora also infers dedicated per-individual bootstrap support values for all individuals in the dataset. These support values indicate whether an individual's location in the lower-dimensional embedding space is stable across bootstrapped embeddings, or, if the individual should be considered as being "rogue" when its projections between distinct bootstrapped embeddings differ substantially. For such a "rogue" individual, any conclusion concerning, for instance, the assignment to a specific (sub-)population should be carefully (re-)considered.

Pandora is implemented in Python and is available open-source on GitHub `https://github.com/tschuelia/Pandora`. It can be used via the command line or as a Python library. A thorough documentation of the tool, including usage examples, is available at `https://pandorageno.readthedocs.io`.

5.2 Methods

Figure 5.1 outlines our algorithm to estimate the stability of a genotype dataset under dimensionality reduction via bootstrapping. We initially generate B bootstrap replicates by randomly sampling from the set of Single Nucleotide Polymorphisms (SNPs) in the input genotype data *with* replacement. Then, for each bootstrap replicate, we perform a dimensionality reduction using either PCA or MDS, depending on the user's choice. Based on the computed embedding replicates, we subsequently compute three stability values: the overall Pandora stability (PS), the per-individual Pandora Support Values (PSVs), and, after applying k-means clustering, the Pandora Cluster Stability (PCS).

In the following, we describe Pandora's bootstrap procedure in more detail, focusing on the pairwise comparison of two embeddings, as well as the computation of the

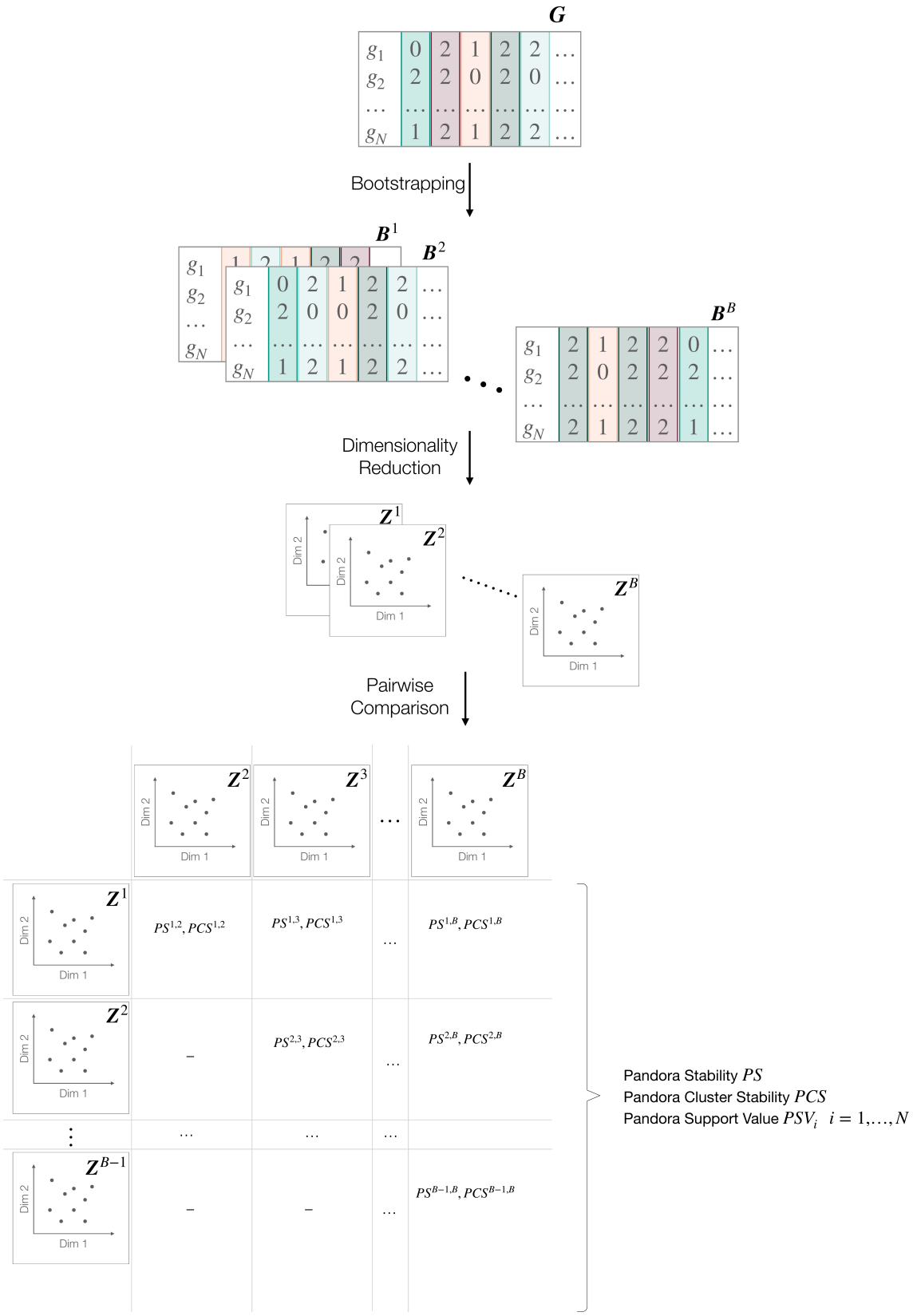


Figure 5.1: Schematic overview of the bootstrap-based stability analyses for genotype data, as implemented in our Pandora tool.

PS, PCS, and PSVs values. For further details on genotype data and dimensionality reduction, see Section 2.2.

Henceforth, the term 'population' refers to its biological rather than to its statistical interpretation. We refer to a genotype data sequence in the input data as a single individual and, in general, each individual is associated to a population (for instance "Greek").

5.2.1 Genotype Data Representation

The input to Pandora is a genotype data matrix $\boldsymbol{G} \in N \times M$ comprising M SNPs recorded for N individuals (see Section 2.2.1). Pandora supports two distinct genotype data representations: file-based and *NumPy*-based. For the file-based input, Pandora expects three distinct files per dataset: one genotype file containing the sequence data, one file containing metadata for each SNP, as well as one file containing metadata for each individual. Pandora supports the three file formats defined by the EIGENSOFT software package [133] (*EIGENSTRAT*, *ANCESTRYMAP*, *PACKEDANCESTRYMAP*), as well as the *pedigree*, and *binary pedigree* formats defined by the PLINK software package [143]. The *NumPy*-based representation relies on data matrices as defined by the Python library *NumPy* [69]. This allows users to preprocess their dataset using custom (Python) scripts without requiring an explicit (and potentially complicated as well as error-prone) conversion into a specific population genetics file format. Additionally, this input format allows for a more generic deployment of Pandora in other fields of biology beyond population genetics, such as PCA analyses of gene expression data, for instance.

Note that Pandora requires a preprocessed dataset, as it does not internally handle preprocessing. Thus, preprocessing steps, such as pruning Linkage Disequilibrium (LD) or rare variant filtering, must be completed *before* conducting Pandora analyses. This prerequisite does not constitute a limitation of Pandora as it applies to any PCA or MDS analysis. For instance, EIGENSOFT's *smartpca* tool also expects a preprocessed dataset when performing a PCA analysis. Thus, regardless of the specific analytical tool employed, conducting proper dataset preprocessing constitutes a fundamental step in population genetics research.

5.2.2 Dimensionality Reduction

Pandora supports two distinct dimensionality reduction techniques: Principal Component Analysis (PCA; see Section 2.2.2.1) and Multidimensional Scaling (MDS; see Section 2.2.2.2).

If the user provides the genotype data using the file-based representation, Pandora computes all PCA embeddings via EIGENSOFT's *smartpca* tool, as it has been specifically designed and highly optimized for population genetics data [133, 140]. Therefore, Pandora supports and propagates all additional *smartpca* PCA analysis commands, including the option to project (ancient) individuals onto an embedding computed using another, distinct set of individuals. For MDS analyses, Pandora

computes the F_{ST} distance matrix using *smartpca* and subsequently applies metric MDS as implemented in *scikit-allel* [120].

Using the alternative *NumPy*-based data representation, Pandora performs all PCA analyses using the PCA implementation provided in the Python machine learning library *scikit-learn* [135]. In analogy to the file-based data representation, the default distance metric is the F_{ST} distance (in this case as implemented in *scikit-allel*). Pandora also provides alternative distance metrics such as the Euclidean, Manhattan, or Hamming distances. The subsequent MDS is computed using the *scikit-allel* implementation of metric MDS.

Note that choosing the appropriate number of components for PCA or MDS constitutes a challenging and debated question. Patterson *et al.* [133] propose to base this decision on the Tracy-Widom statistic, but Elhaik [41] claims that this statistic is too sensitive and results in overestimating the appropriate number of PCs. A common criterion used in other PCA application domains is to compute as many PCs as required to explain, for instance, 80% of the variance. However, this is not applicable to genotype data due to its extremely high dimensionality. For example, to explain 80% of the variance for a dataset analyzed by Lazaridis *et al.* [102] comprising approximately 600 000 SNPs, over 500 PCs need to be computed. In analogy to Price *et al.* [140], we thus set the default number of components for PCA analyses in Pandora to 10. For MDS analyses, we set the default number of components to 2. Pandora users can specify any number of components that best suit their analysis. Pandora uses all computed PCA or MDS components for its stability analyses. Further, note that the user needs to make an informed decision on the number of components to perform dimensionality reduction regardless of using Pandora.

To keep the following description of Pandora as generic as possible, we henceforth refer to the result of a PCA or MDS analysis as *embedding* and denote it as $\boldsymbol{Z}$.

5.2.3 Bootstrapping Genotype Data

The bootstrap procedure [40] is a statistical approach to quantify measures of accuracy based on random sampling with replacement from a set of observations. Estimating the stability of PCA using bootstrapping is not a novel concept. For example, Fisher *et al.* [50] propose a bootstrap-based method to estimate the sampling variability of PCA results on datasets where the number of features is substantially larger than the number of samples. However, standard bootstrapping methods are not directly applicable to genotype data. A necessary condition for bootstrapping is the (assumed) independence of the bootstrapped dimensions. In general, individuals (especially individuals of the same species) do not evolve independently of each other. However, Felsenstein [48] argues that we may assume independent evolution of genetic loci. Therefore, in Pandora, instead of sampling the individuals, we sample the SNPs to obtain a new bootstrap replicate matrix $\boldsymbol{B} \in N \times M$. Thus, a bootstrap replicate has the same set of individuals but a distinct SNP composition. In doing so, we follow the standard approach used in phylogenetics, as first proposed by Felsenstein [48].

The computations of embeddings for individual bootstrap replicates are independent of each other, and their computations are therefore straight-forward to parallelize. Pandora exploits this parallelism and also allows users to specify the number of threads to use.

Since Pandora computes a lower-dimensional embedding (a compute-intensive dimensionality reduction) of each bootstrap replicate, the computational resource requirements for Pandora's uncertainty estimation are substantial. To alleviate this, we periodically perform a heuristic convergence assessment until a maximum number of bootstrap replicates is reached. The frequency of this convergence check is determined by the number of threads being used. During the convergence check, Pandora assesses the variation of PS estimates based on random subsets of the computed replicates at the time of the convergence check. When this variation falls below a certain tolerance level, Pandora terminates all remaining bootstrap computations and conducts the final stability value calculation using the set of completed bootstrap replicates. We provide a more detailed description of our convergence criterion in Section 5.2.5 below.

5.2.4 Stability Estimation

5.2.4.1 Pandora Stability

The Pandora Stability (PS) describes the overall stability of the genotype dataset that is based on the pairwise similarity scores over all bootstrap replicates. To compute the similarity between two bootstrap embeddings $\boldsymbol{Z}^u$ and $\boldsymbol{Z}^v$, we follow the approach suggested by Wang *et al.* [184] using Procrustes analysis. Procrustes analysis determines the optimal transformation $f(\boldsymbol{Z}^u)$ that matches the projections of all individuals $\boldsymbol{g}_i$ in $\boldsymbol{Z}^u$ to $\boldsymbol{Z}^v$ as accurately as possible while preserving the relative distances between individuals. This transformation consists of a scaling factor, a rotation and reflection matrix, and a translation matrix. In Pandora, we use the Procrustes analysis as implemented in the Python scientific computing package *SciPy* [182]. Before computing the optimal transformation, both embeddings are standardized such that $trace(\boldsymbol{Z}\boldsymbol{Z}^\top) = 1$, and both sets of points are centered around the origin to remove translation effects. The optimal transformation minimizes the squared Frobenius norm (also called disparity)

$$D^{u,v} = \|f(\boldsymbol{Z}^u) - \boldsymbol{Z}^v\|_F^2. \tag{5.1}$$

The disparity $D^{u,v}$ has a minimum of 0 and a maximum of 1 [184]. Based on this *disparity*, we can compute the pairwise *similarity* as

$$PS^{u,v} = \sqrt{1 - D^{u,v}} \tag{5.2}$$

on a scale from 0 to 1, with higher values indicating higher similarity [184].

To obtain the overall PS, we average over the pairwise similarity across all $\frac{B(B-1)}{2}$ possible distinct pairs of bootstrap replicates

$$PS = \frac{2}{B(B-1)} \sum_{\substack{1 \le u,v \le B \\ u<v}} PS^{u,v}. \tag{5.3}$$

The resulting PS is again a value between 0 and 1. The higher the PS, the more similar the bootstrap replicates are, with 1 indicating that all bootstrap embeddings identically project all individuals.

5.2.4.2 Pandora Cluster Stability

In analogy to the PS, we initially compute the pairwise Pandora Cluster Stability (PCS) for all unique pairs of bootstrap embeddings and then average over all pairwise scores to obtain the overall PCS

$$PCS = \frac{2}{B(B-1)} \sum_{\substack{1 \leq u,v \leq B \\ u<v}} PCS^{u,v}. \tag{5.4}$$

To compute the pairwise $PCS^{u,v}$ for bootstrap embeddings $\boldsymbol{Z}^u$ and $\boldsymbol{Z}^v$, we first match both embeddings using Procrustes analysis as described above and subsequently perform an independent k-means clustering for the two embeddings. To compare the resulting label assignments for all individuals, we use the same number of clusters k for both embeddings. Finally, we compute the pairwise $PCS^{u,v}$ for bootstrap embeddings $\boldsymbol{Z}^u$ and $\boldsymbol{Z}^v$ via the Fowlkes-Mallows index [56].

The PCS is a value between 0 and 1. The higher the PCS, the higher is the similarity between clustering results across bootstraps.

Selecting the number of clusters k to best represent the dataset is not trivial. Pandora users can manually set the number of clusters based on, for instance, prior knowledge of the expected underlying (sub-)population structure in the data. When the user does not explicitly specify k, we automatically determine the optimal k based on a grid search and the Bayesian Information Criterion (BIC) [154]. To reduce the computational cost of searching for the optimal k, we determine an input data specific upper bound via the following heuristic. If the individuals of the dataset to be analyzed include population annotations, we set the upper bound to the number of distinct populations in the dataset. If this information is not available, we set the upper bound to the square root of the number of individuals in the dataset. Based on empirical parameter exploration experiments, we set the lower bound to 3. During Pandora development, we observed a substantial influence of k on the resulting PCS scores for various empirical population genetics datasets. Figure 5.2 depicts this observation for a dataset of West-Eurasian individuals [102] with 100 bootstrap replicates for $k \in [3, 15]$. While we do observe a general trend for decreasing PCS values with an increasing number of clusters k, we also observe fluctuations for $k \in [5, 8]$ without a clear trend.

We thus strongly recommend users to provide a reasonable k to Pandora based on prior knowledge or preliminary experiments, for instance, using the number of distinct populations in the dataset. We further note that the PCS should only be reported in conjunction with the number of clusters k.

Figure 5.2: PCS for the *HO-WE* dataset as a function of the number of clusters k used for k-means clustering.

5.2.4.3 Pandora Support Values

To assess the projection stability for each individual in the dataset, we compute the Pandora support for each individual $\boldsymbol{g}_i$ as

$$PSV_i = 1 - \frac{\sum_{\substack{1 \leq u,v \leq B \\ u<v}} \|\boldsymbol{g}_{i,u} - \boldsymbol{g}_{i,v}\|}{2B \cdot \sum_{1 \leq u \leq B} \|\boldsymbol{g}_{i,u}\|} \tag{5.5}$$

where $\boldsymbol{g}_{i,u}$ and $\boldsymbol{g}_{i,v}$ denote the projection of $\boldsymbol{g}_i$ in bootstrap u and v, respectively.

The fraction in Equation (5.5) corresponds to the Gini coefficient, a statistical measure of dispersion from economics. In our case, the Gini coefficient measures the dispersion of the projections of an individual with respect to all bootstrap replicates on a scale from 0 to 1. The higher the dispersion, the higher the Gini coefficient will be. To align this with the interpretation of the PS and PCS values, where 0 corresponds to unstable and 1 to stable, we simply subtract the calculated dispersion measure from 1.

5.2.5 Bootstrap Convergence Criterion

As stated above, the implemented bootstrap procedure in Pandora induces high computational resource requirements. To reduce this computational overhead, we periodically perform a heuristic convergence assessment until a maximum number of bootstrap replicates is reached. The frequency of this convergence check is determined by the number of threads used.

The default setting is a maximum of $B \coloneqq 100$ bootstrap replicates. Now, let B^* be the number of bootstrap replicates already computed when performing the convergence assessment. We create 10 subsets of size $\tilde{B} = \frac{B^*}{2}$ by sampling 10 times

without replacement from the set of B^* completed bootstraps. We then compute the Pandora Stability (PS) for each of the 10 subsets and determine the relative difference of PS values between all pairs of PS values (PS^u, PS^v): $\frac{|PS^u - PS^v|}{PS^u}$. We assume convergence if no pairwise relative difference exceeds α. In other words, we require that the highest sampled stability is at most $\alpha \cdot 100\%$ larger than the lowest sampled stability. If this is the case, all remaining bootstrap computations are omitted. Pandora allows users to explicitly set the convergence tolerance α; the default setting is 0.05. Note that we decided to sample 10 subsets of size $\tilde{B}$ as a trade-off between the accuracy and the additional runtime overhead induced by the bootstrap convergence assessment. The runtime overhead induced by the convergence assessment is substantial, as we need to compute the PS for each of the subsets, meaning that we need to perform $\frac{\tilde{B}(\tilde{B}-1)}{2}$ Procrustes analyses.

We also determine the frequency of invocation of this convergence assessment based on the number of threads specified by the user, which defines the size of a full batch of bootstrap replicates. Pandora checks for convergence after every full batch or once 10 replicates are computed, whichever number is higher. For example, if a user executes Pandora with 40 threads, it will compute 40 bootstrap replicates in parallel and hence complete 40 replicates at approximately the same time. If Pandora were to run the convergence assessment after completing 10 bootstraps and determined that it has converged, we would discard 30 (almost) finished bootstraps, thus wasting the resources used for their computation, as well as for three unnecessary convergence assessment computations (after 10, 20, and 30 completed replicates).

5.2.6 Simulating Genotype Data

We use simulated genotype data to verify whether Pandora can detect instability in genotype data under dimensionality reduction. We simulated genotype datasets using the *stdpopsim* python library [3, 101]. This library provides a catalog of 13 distinct, published human demographic models describing the demographic history of various human populations. We simulated genotype data according to each model to generate realistic whole-genome population genetic data. Note that we use *msprime* [14] as simulation engine for all following *stdpopsim* simulations. The *msprime* tool simulates data via the coalescent model (see Section 2.2.1.1). To not bias the simulations in favor of single populations, we simulated the same number of individuals for each population in the demographic model, up to a total of 500 individuals.

Figure 5.3 visualizes our simulation pipeline. We first simulated genotype datasets with a *target* sequence length of 10^5, 10^6, 10^7, and 10^8 nucleotides. While the resulting simulated sequences are of this target length, not all positions in the simulated sequences are SNPs. Consequently, we only used the SNPs for the subsequent analyses. Following standard population genetics data preprocessing procedures [7, 102], we filtered rare variants (*MAF filtering*) and removed correlated SNPs by applying Linkage Disequilibrium (LD) pruning using PLINK 2.0 [29]. We removed SNPs with an allele frequency below 1%. We applied LD pruning with a r^2 threshold of 0.5, a

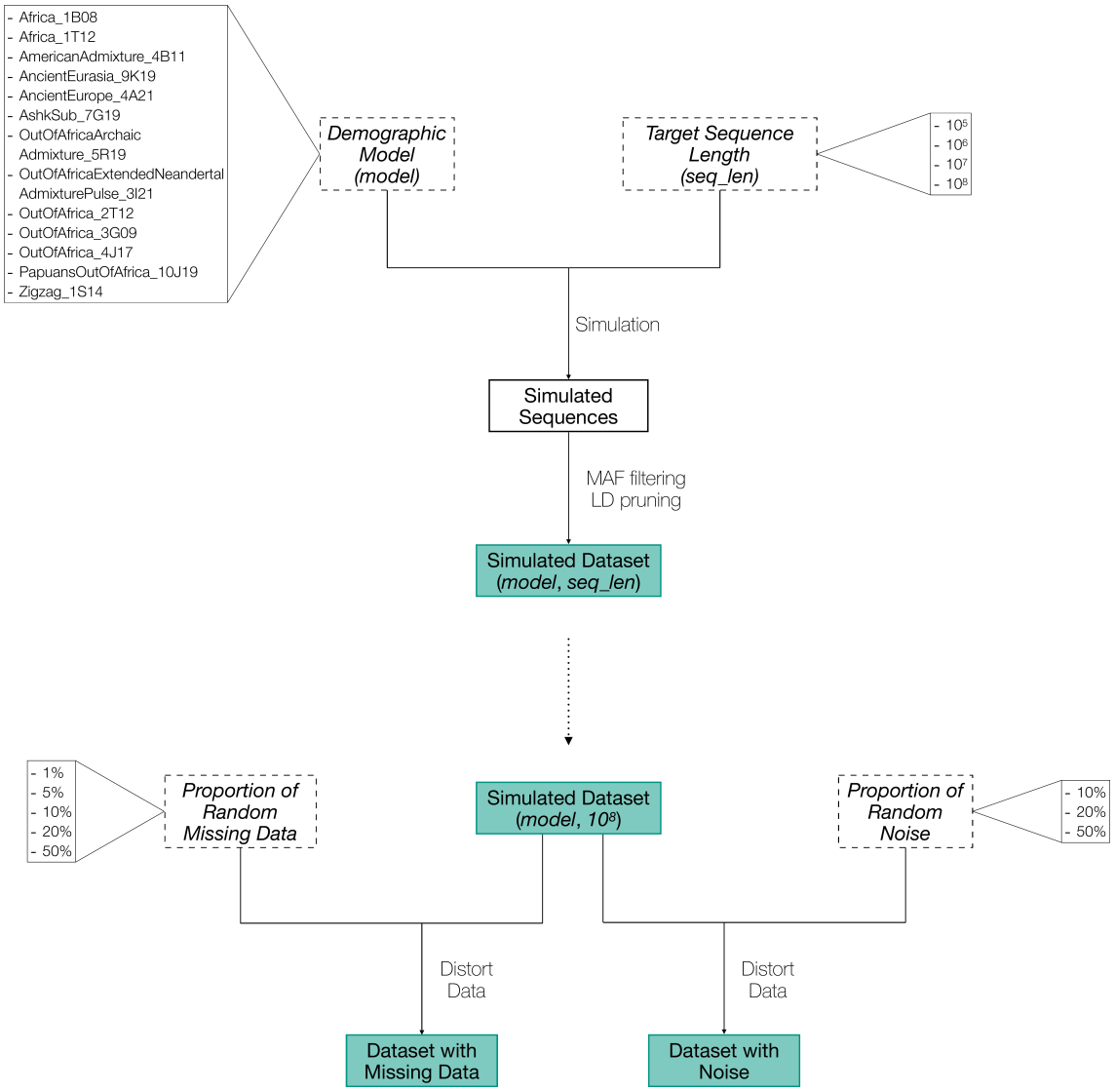


Figure 5.3: Schematic overview of the implemented simulation pipeline.

window size of 50 and a stride of 5. Table 5.1 states the resulting number of SNPs per simulated dataset (by model and target sequence length).

Using our 13 simulated datasets with a target sequence length of 10^8, we distorted the data by injecting random missing data and random noise. For the missing data simulations, we randomly replaced a certain proportion of entries in the genotype matrix with the missing value character (1%, 5%, 10%, 20%, and 50%). For simulating noisy data, we randomly replaced a certain proportion of values in the genotype matrix with random variants (10%, 20%, and 50%). The number of SNPs remains unaffected by these changes. Note that we did not mix missing and noisy data simulations. That is, for the missing data simulations we did not inject additional noise, and for the noise data simulation we did not inject missing data.

We performed Pandora stability analyses using PCA and MDS for all datasets highlighted in Figure 5.3. For PCA analyses, we reduced the data to 10 dimensions (the

Demographic Model	Target Sequence Length			
	10^5	10^6	10^7	10^8
Africa_1B08	217	2255	22 172	222 290
Africa_1T12	195	1924	18 924	190 410
AmericanAdmixture_4B11	165	1470	13 076	135 622
AncientEurasia_9K19	142	1331	13 446	133 883
AncientEurope_4A21	60	692	6722	65 006
AshkSub_7G19	183	2202	21 573	207 071
OutOfAfrica-ArchaicAdmixture_5R19	105	1019	10 546	106 972
OutOfAfricaExtended-NeandertalAdmixturePulse_3I21	272	2920	29 971	296 220
OutOfAfrica_2T12	182	1769	16 721	163 492
OutOfAfrica_3G09	153	1560	15 725	156 819
OutOfAfrica_4J17	127	1502	14 477	142 285
PapuansOutOfAfrica_10J19	411	3994	38 065	373 325
Zigzag_1S14	150	1586	17 449	171 090

Table 5.1: Number of SNPs in the simulated population genetics datasets per target sequence length.

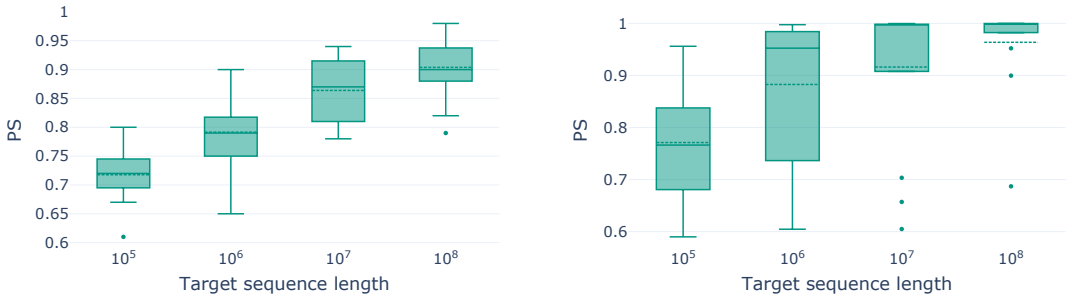
default setting in *smartpca*). For MDS analyses, we reduced the data to 2 dimensions, and we used the Euclidean sample distance as the distance metric. We used 20 threads for all Pandora analyses. If not stated otherwise, we used Pandora's default bootstrap convergence setting (convergence tolerance of 5%).

We implemented the outlined simulation pipeline using the *Snakemake* workflow management system [94]. Note that we explicitly set the random seed for each simulation to ensure reproducible simulation results. We provide all scripts required for reproducing our simulations on GitHub at `https://github.com/tschuelia/PandoraPaper`, and all simulated datasets (including all datasets with random missing and noise data) in EIGENSTRAT format at `https://cme.h-its.org/exelixis/material/Pandora_supplementary_data.tar.gz`.

5.3 Results

In the following, we demonstrate the utility of Pandora. Due to the novelty of our approach, and the novelty of stability estimation of genotype data under dimensionality reduction, there exists no benchmark data collection to compare Pandora against. We therefore verify and demonstrate the functionality of our proposed approach using our simulated genotype data and empirical, published population genetics datasets. We additionally analyze the influence of the implemented bootstrap convergence check on the stability estimates and the runtime of Pandora.

All scripts to reproduce the following analyses, as well as all results, are available via the aforementioned GitHub and data download links.



(a) PS values for PCA analyses.

(b) PS values for MDS analyses.

Figure 5.4: PS values for PCA (left panel) and MDS (right panel) analyses as a function of the target sequence length for the simulated genotype datasets.

5.3.1 Simulated Data

We first analyze the simulated data with varying sequence lengths (10^5, ..., 10^8). Simulating longer genomes increases the number of SNPs in the simulated dataset. With increasing SNP numbers, we expect the amount of informative data in the genotype dataset to also increase. Thus, we expect dimensionality reduction to better capture population structure. Consequently, we expect PCA or MDS results to become more stable with increasing SNP numbers in the dataset. Note that, as stated above, while we simulate sequences of length 10^5, this will not necessarily yield 10^5 SNPs since not all genomic sites evolve. Since the resulting SNP number varies as a function of the demographic model, in the following, we identify datasets via the *target* sequence length. Figure 5.4 shows the distribution of PS values for PCA (left panel) and MDS (right panel) analyses as a function of the target sequence length. The expected trend of higher stability with increasing target sequence lengths is clearly visible for both, PCA, and MDS. This trend can be further visualized by plotting the first two principal components of the PCA analysis per target sequence length. Figure 5.5 visualizes the first two principal components of the PCA computed for data simulated under the AncientEurope_4A21 demographic model with increasing target sequence lengths (10^5, ..., 10^8). The longer the target sequence, the better the population structure becomes visible. As expected, the PS increases with increasing sequence lengths.

Missing data is known to affect the stability of dimensionality reduction [196]. We therefore expect decreasing PS values with increasing proportions of missing data. To test this hypothesis, we distorted simulated datasets by injecting missing data into our datasets with a target sequence length 10^8. Note that we used the datasets with the maximum target sequence length to not bias the following analysis by weak(er) signal resulting from small SNP numbers in the dataset. For the missing data impact analysis, we distorted the dataset by replacing proportions of the data (1%, 5%, 10%, 20%, and 50%) with the missing data character. Note that this does not alter the number of SNPs per dataset. Further, note that imputing (or

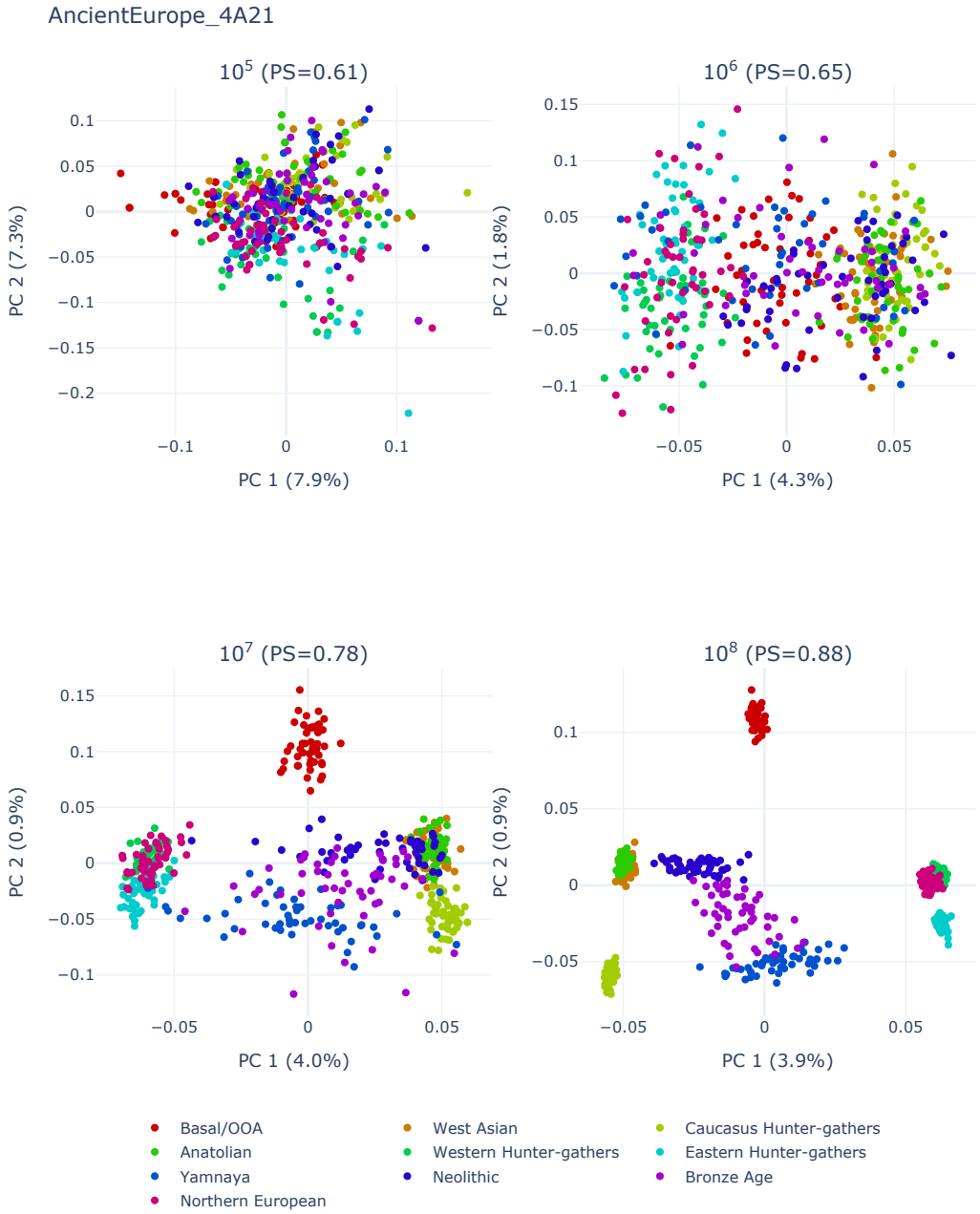
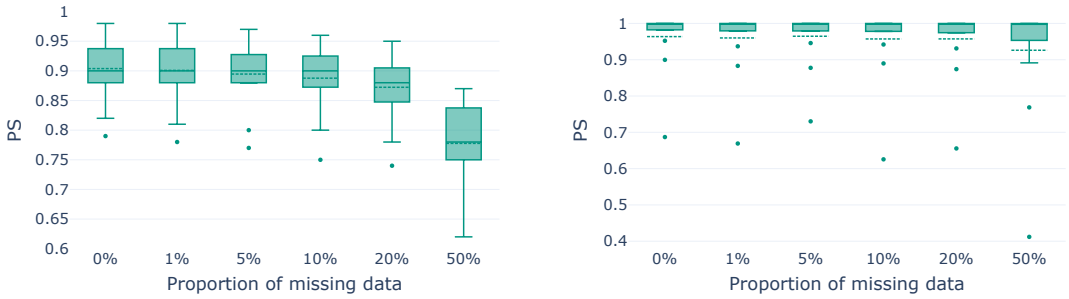


Figure 5.5: The first two principal components of the PCA computed for data simulated under the AncientEurope_4A21 demographic model with increasing target sequence lengths.

removing) missing data is essential for PCA or MDS analyses, as the underlying mathematical frameworks cannot process incomplete datasets. Thus, we imputed missing data using mean imputation.

Figure 5.6(a) shows the distribution of PS values as a function of the missing data proportion for PCA analyses. We do observe the expected trend of decreasing PS



(a) PS values for PCA analyses.

(b) PS values for MDS analyses.

Figure 5.6: PS values for PCA (left panel) and MDS (right panel) analyses as a function of the proportion of random missing data for the simulated genotype datasets.

values with increasing missingness. Yet, even with 50% missing data, the results are relatively stable with an average PS value of 0.78, and its decline is not as substantial as one might expect. One reason for this is that the simulated datasets are overall clearly structured with highly distinct populations. For example, Figure 5.7 shows the first two principal components of the dataset simulated according to the demographic model of ancient European populations with increasing proportions of missing data. Despite the high proportion of missing data of 50% (bottom right panel), the population structure was easily recovered. We observe only a slight dispersion within the individual population clusters compared to the PCA on the dataset without missing data (upper left panel). Furthermore, Yi and Latch [196] report analogous findings: while PCA results are affected by random missing data, high proportions of random missing data can be compensated by a strong underlying population structure.

For MDS analyses, we observe no substantial effect of missing data on the stability of genotype datasets for missing data proportions of up to 20%. Even for 50% missing data, we only observe a slight PS decrease (Figure 5.6(b)). Evidently, the distance matrix calculation is comparatively robust to missing data imputation, as is the MDS embedding construction with respect to small changes in the distance matrix.

In contrast to *missing* data, *noisy* data cannot be imputed, as we do not know which part of the data is noise and which part is "real" data. We expect lower PS values for higher proportions of noise in the dataset because uninformative noise distorts the dataset structure. For the following simulations, we used our simulated datasets with a target sequence length of 10^8 and without any missing data. We distorted the data by adding 10%, 20%, and 50% of noise. To this end, we altered single genotypes of randomly selected SNPs and individuals in the genotype matrix. As expected, we observe a substantial stability decline with increasing proportions of random noise. We observe this result for both, PCA (Figure 5.8(a)), and MDS (Figure 5.8(b)) analyses. However, the effect is substantially more pronounced in

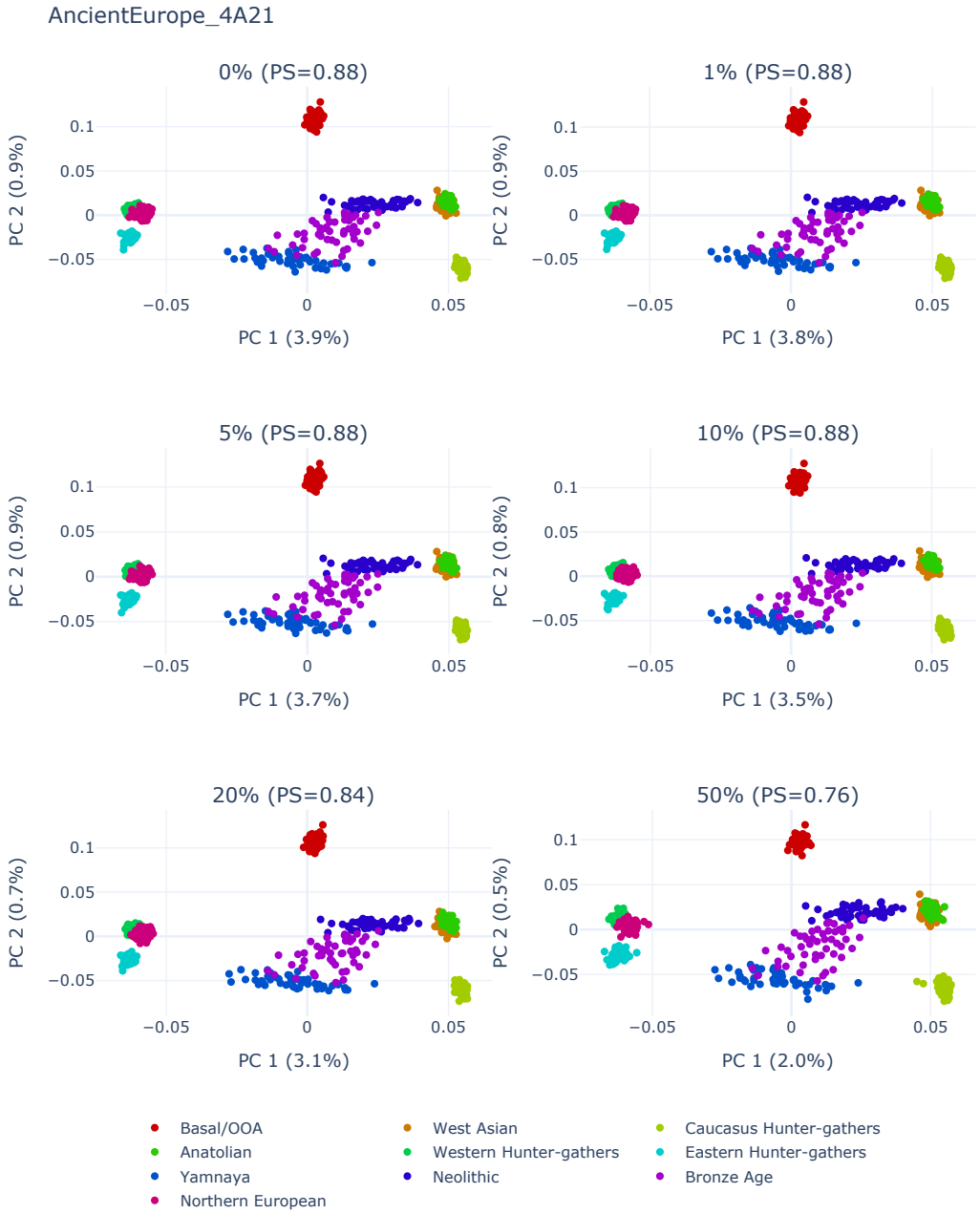
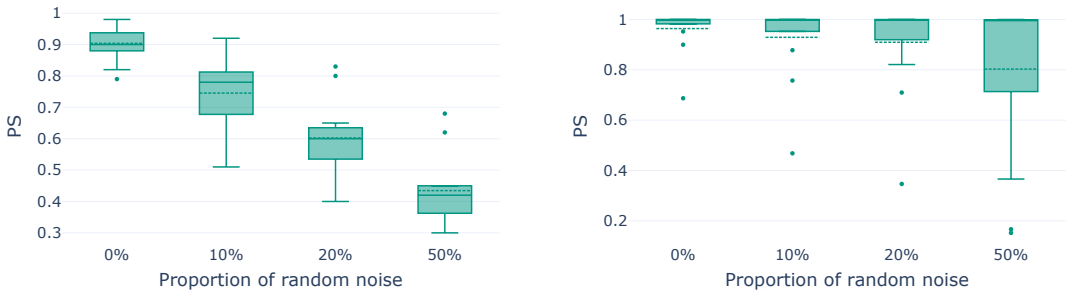


Figure 5.7: The first two principal components of the PCA computed for data simulated under the AncientEurope_4A21 demographic model with a target sequence length of 10^8 and increasing proportions of random missing data.

PCA analyses. In analogy to the argument above, we suspect MDS to be less prone to distorted data due to the distance matrix-based approach.

Using the simulated dataset of the demographic model of ancient European individuals, Figure 5.9 visualizes the impact of noise on the resulting population structure in



(a) PS values for PCA analyses.

(b) PS values for MDS analyses.

Figure 5.8: PS values for PCA (left panel) and MDS (right panel) analyses as a function of the proportion of random noise data for the simulated genotype datasets.

the PCA, clearly showing a higher dispersion of individuals with higher proportions of noise.

Given that Pandora consistently reflects the expected trends of increasing stability with an increasing number of SNPs, and decreasing stability with increasing proportions of missing and noisy data, we conclude that bootstrapping-based stability estimation adequately captures instability in dimensionality reduction studies.

5.3.2 Empirical Data

All simulated datasets exhibit a clear underlying population structure. As this is not necessarily the case for empirical data, we also demonstrate the utility of Pandora on empirical datasets. Additionally, PCA or MDS studies are frequently applied in ancient DNA (aDNA) studies. Usually, in aDNA studies, the embedding is computed using a dataset of modern individuals. Subsequently, ancient individuals are projected onto the resulting PCA or MDS embedding. In human population studies, especially in work on newly sequenced ancient specimen, the Human Origins (HO) SNP Array [102] comprising 2068 publicly available individuals is used as a reference dataset of modern individuals. For our first empirical analysis, we assembled the West-Eurasian subset based on the description provided by Lazaridis *et al.* [102]. The resulting West-Eurasian subset contains 813 individuals from 55 distinct populations. We henceforth denote this dataset as *HO-WE*. The *HO-WE* dataset contains 605 775 SNPs with 0.5% missing data. We then merged the *HO-WE* dataset with the 13 ancient Çayönü individuals published by Altınışık *et al.* [7] (*HO-WE-Çayönü*). These ancient individuals contain between 90% and 99% of missing data. Using the *smartpca* option `lsqproject:YES`, we computed all PCA embeddings in the following analysis using the modern individuals only and then projected the 13 ancient Çayönü individuals onto the resulting embeddings. According to Pandora, the *HO-WE-Çayönü* dataset is overall very stable under PCA, with a PS of 0.89. In their analyses, Altınışık *et al.* [7] computed a 95% confidence region for the first two principal components using the confidence ellipse function in *smartpca*. This

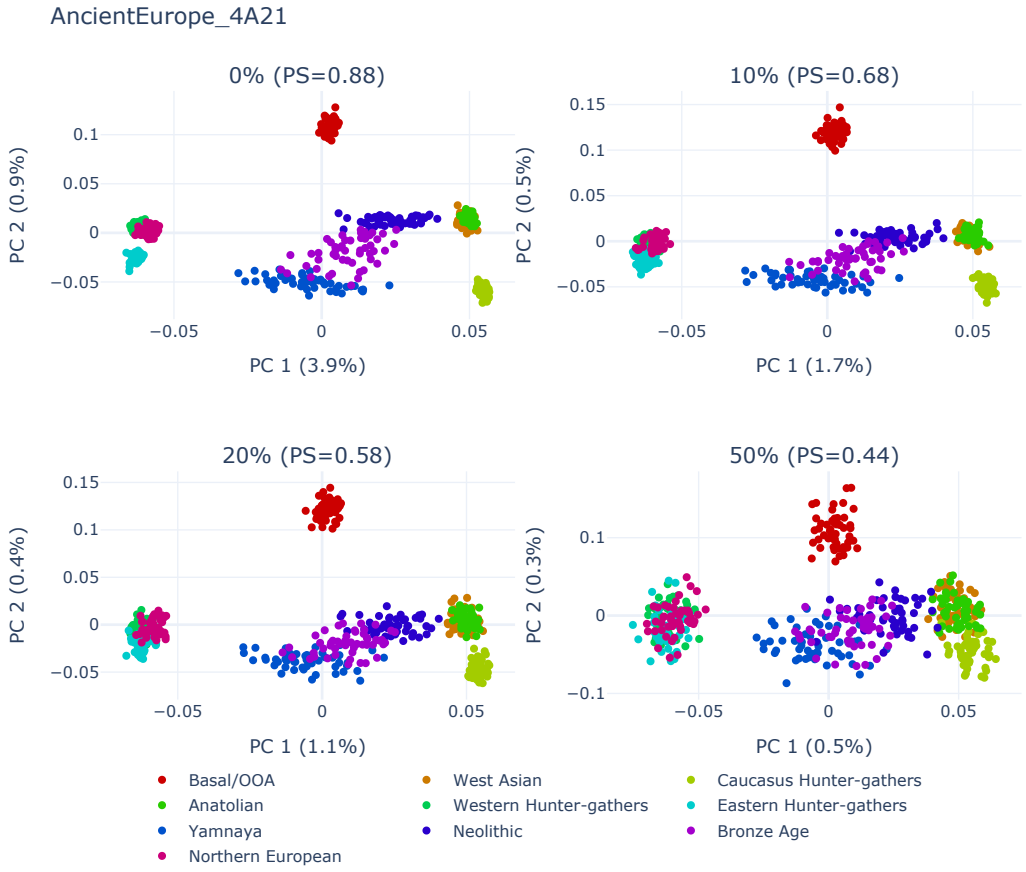


Figure 5.9: The first two principal components of the PCA computed for data simulated under the AncientEurope_4A21 demographic model with a target sequence length of 10^8 and increasing proportions of random noise.

confidence interval indicates a high projection uncertainty, especially for the *cay015* individual. Our results are in line with these findings, as the *cay015* individual has a PSV of only 0.54 according to our analyses. Figure 5.10 visualizes this behavior, showing the computed PCA embeddings of two bootstrap replicate datasets and highlighting the *cay015* individual. A comparison of the figures clearly shows that the positions of the *cay015* individual in both embeddings differ substantially, hence the low PSV. For individuals with such low PSVs, conclusions based on PCA analyses should only be drawn with extreme caution. For example, *cay015* is projected closest to an Israeli individual in the PCA of the unbootstrapped *HO-WE-Çayönü* dataset (*BedouinA* population). However, when considering the two bootstraps we used in Figure 5.10, *cay015* is projected closest to a Yemeni individual (bootstrap replicate #1; *Yemenite-Jew* population) and a Libyan individual (bootstrap replicate #2; *Libyan-Jew* population), respectively. These observations are again in line with the findings of Altınışık *et al.* [7], as they conclude that Çayönü is a genetically diverse population.

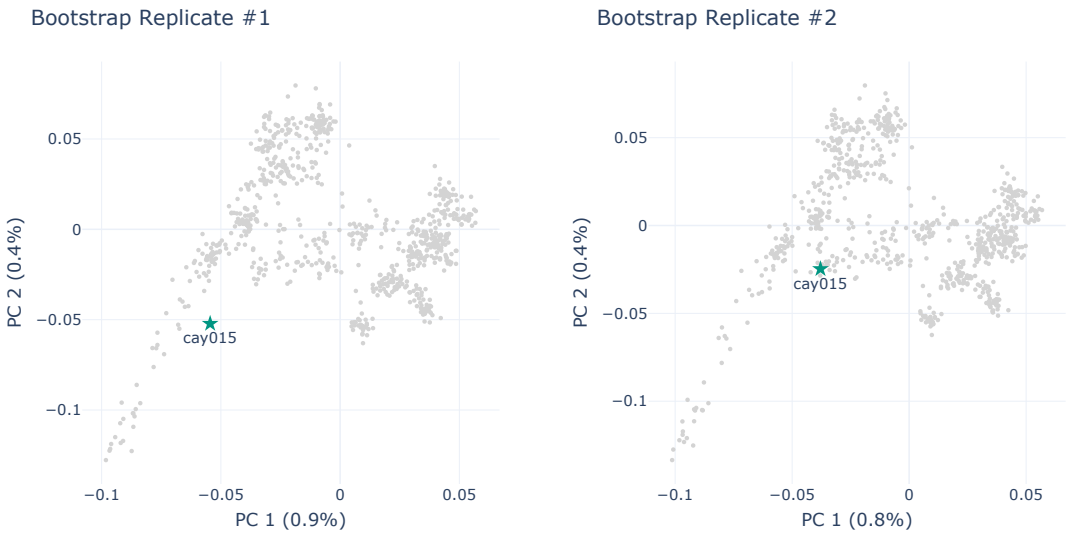


Figure 5.10: PCA embeddings of two bootstrap replicates of the *HO-WE-Çayönü* dataset The highlighted points correspond to the respective projection of the *cay015* individual.

We suggest filtering individuals with low PSVs before conducting any additional analyses. If, however, the individuals with low PSVs are important to the study, we suggest performing additional analyses, for instance admixture analyses. Since version 8, *smartpca* includes a confidence ellipse functionality that plots a 95% confidence region for individuals. This functionality can be particularly useful for placing ancient individuals (see for example the analyses in Altınışık *et al.* [7]). However, the confidence ellipse only considers the first two principal components and does not *quantify* the instability. Instead, it yields a plot showing the confidence regions. Consequently, this approach cannot be applied simultaneously to all individuals in the dataset, as the resulting figure would be overly complicated and uninformative. With Pandora, users can pre-filter unstable individuals with low PSVs and subsequently perform a more targeted confidence region analysis.

Finally, we analyzed two datasets of genomic dog data published by Morrill *et al.* [126]. Using the respective definition of Morrill *et al.* [126], we assembled a subset of 601 *purebred* dogs and a subset of 1071 *highly admixed* dogs based on the 2155 publicly available genomic dog sequences comprising 6 059 222 SNPs. The idea of this analysis was to demonstrate the usage and utility of Pandora's PCS estimate. We determined the number of distinct breeds in each dataset using the data provided by Morrill *et al.* [126]. The purebred dogs comprise 88 distinct breeds, whereas the highly admixed dogs comprise 60 distinct dog breeds. We assessed the stability of breed assignments via the PCS as estimated by Pandora, using the respective number of breeds as k in k-means clustering. Morrill *et al.* [126] consider a dog as being *highly admixed* if it has below 45% ancestry from any single breed in admixture analysis. Consequently, we expect a lower PCS for the *highly admixed* dataset compared to the *purebred* dataset. In fact, for the *highly admixed* dataset we observe a PCS of

only 0.63, whereas the *purebred* dataset is substantially more stable under k-means clustering with a PCS of 0.87.

5.3.3 Speedup and Accuracy under the Bootstrap Convergence Criterion

In the following, we describe our analysis setup to assess the influence of the implemented bootstrap convergence check as described in Section 5.2.5 on the stability estimates and the runtime of Pandora. For each of the 13 simulated genotype datasets (target sequence length 10^8 and no missing or noisy data), we performed the following analyses. As baseline, we executed Pandora with the full set of $B_{100} := 100$ bootstrap replicates (default Pandora setting) and disabled the bootstrap convergence check. We denote the runtime of this execution by T_{100}, the PS as PS_{100}, and the PSVs as PSV_{100}. We further executed Pandora twice with the convergence check enabled, but with a different convergence tolerance in each execution. In one execution, we set the convergence tolerance to 0.05 (5%) and in a second execution to a more conservative value of 0.01 (1%). We denote the results of both executions as $T_{5\%}, PS_{5\%}, PSV_{5\%}$ and $T_{1\%}, PS_{1\%}, PSV_{1\%}$, respectively. We further report the number of bootstraps required for convergence in both runs ($B_{5\%}$, $B_{1\%}$). To compare the runtimes, we computed the speedups $S_{5\%} := \frac{T_{100}}{T_{5\%}}$ and $S_{1\%} := \frac{T_{100}}{T_{1\%}}$. A speedup below 1 indicates that the runtime with the convergence check enabled exceeds the runtime of computing all 100 replicates without checking for convergence. In cases where the bootstrap procedure does not converge before all 100 bootstraps have been computed, the speedup is necessarily below 1 due to the additional convergence checks. To analyze the potential loss in stability analysis accuracy induced by the early termination, we computed the deviation of PS values,

$$|PS_{5\%} - PS_{100}| \quad \text{and} \quad |PS_{1\%} - PS_{100}|,$$

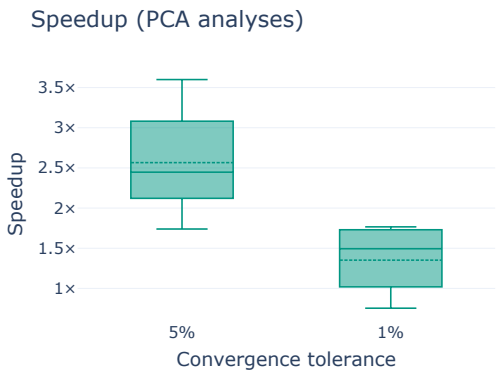
as well as the deviation of PSVs,

$$|PSV^i_{5\%} - PSV^i_{100}| \quad \text{and} \quad |PSV^i_{1\%} - PSV^i_{100}|$$

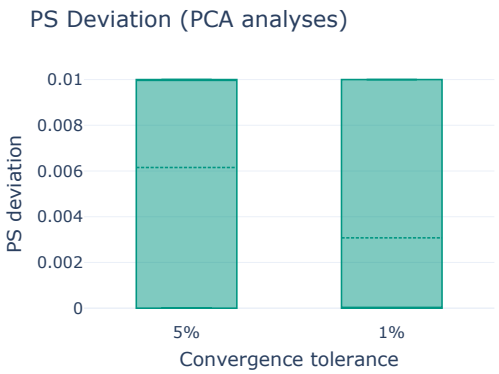
for $i = 1, \ldots, N$.

Figure 5.11 shows box plots for the speedup, PS deviation, and PSV deviation of the Pandora execution under the 5% and 1% convergence tolerance settings for PCA analyses. We observe an average speedup of $2.6 \pm 0.6\times$ for the 5% convergence tolerance. The bootstrap procedure converged after computing 20 replicates for all 13 datasets. With the 1% convergence tolerance, we observe an average speedup of $1.4 \pm 0.4\times$ and bootstrap convergence after 63 replicates on average. The PS values deviate only slightly, with a maximum observed deviation of 0.01 under both settings. The PSVs of individuals differ on average by 0.01 with the 5% convergence tolerance and 0.0 (no deviation) with the 1% convergence tolerance. We observe a maximum deviation of PSVs for single individuals of 0.08 and 0.06, respectively.

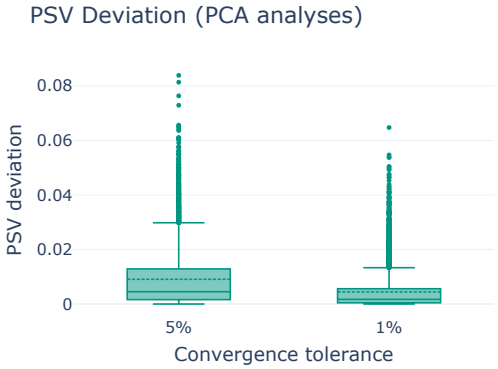
The speedup and deviation of PS values are similar for MDS analyses. We observe an average speedup of $2.2 \pm 0.6\times$ with the 5% convergence tolerance, and an average



(a) Speedup for PCA analyses.



(b) PS deviation for PCA analyses.



(c) PSV deviation for PCA analyses.

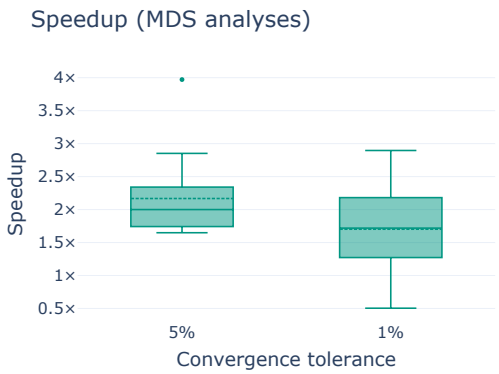
Figure 5.11: Box plots showing the speedup (top left panel), PS deviation (top right panel), and PSV deviation (bottom panel) of the Pandora execution under the 5% and 1% convergence tolerance settings for PCA analyses. Note that we used 20 threads for all Pandora analyses.

speedup of $1.7 \pm 0.7\times$ with the 1% convergence tolerance (Figure 5.12(a)). Pandora converged on average after 20 (5%) and 34 replicates (1%), respectively. In analogy to the PCA results, the PS deviations for MDS deviate only slightly, with maximum deviations of 0.02 under both convergence tolerance settings (Figure 5.12(b)). The average PSV deviation is 0.01 under both settings. However, we observe substantial deviations for single individuals, with differences of up to 0.19 (Figure 5.12(c)).

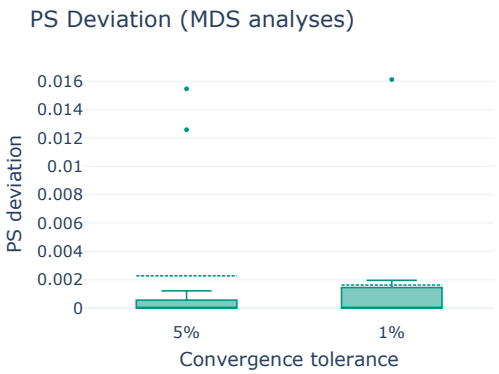
Our results suggest that for PCA and MDS analyses, the 5% convergence tolerance is suitable when the overall PS is of primary interest. When accurate PSVs estimates are paramount, we recommend decreasing the convergence tolerance to 1% for PCA analyses, and disabling the convergence check for MDS analyses.

5.4 Discussion

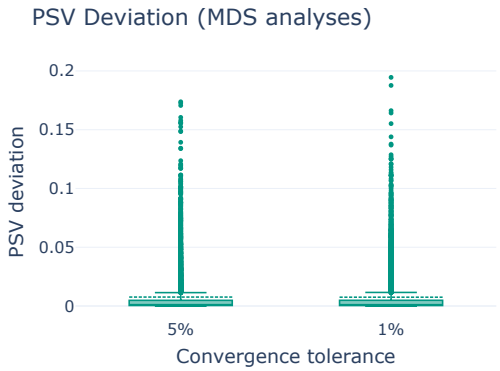
Pandora estimates the stability of a genotype dataset and its individuals under dimensionality reduction via bootstrapping. Pandora supports PCA and MDS analyses. In addition to an overall stability estimate (PS), Pandora applies k-means



(a) Speedup for MDS analyses.



(b) PS deviation for MDS analyses.



(c) PSV deviation for MDS analyses.

Figure 5.12: Box plots showing the speedup (top left panel), PS deviation (top right panel), and PSV deviation (bottom panel) of the Pandora execution under the 5% and 1% convergence tolerance settings for MDS analyses. Note that we used 20 threads for all Pandora analyses.

clustering to all bootstrapped embeddings to estimate the stability of the assigned clusters across all bootstrap replicates (PCS). Furthermore, Pandora computes a bootstrap support value for each individual in the genotype dataset (PSVs). The latter is particularly useful for datasets with projections of ancient individuals onto an embedding computed on modern individuals only. For these analyses, the PSVs for the ancient individuals provide a confidence value for their placement in the embedding space relative to modern individuals. The lower the support for an individual, the more careful users should be when drawing conclusions regarding this individual based on dimensionality reduction analyses.

Our analyses on empirical and simulated data show that Pandora can detect instability in PCA and MDS analyses. As expected, Pandora yields lower stability for both, PCA and MDS analyses, with increasing fractions of missing and noisy data under simulations, as well as higher stability with an increasing number of SNPs in the dataset. Compared to a single PCA or MDS analysis, Pandora introduces

a substantial runtime and resource overhead. Our bootstrap convergence criterion alleviates this to the extent possible.

Our analyses using simulated genotype datasets showed that Pandora is particularly useful when performing PCA or MDS analyses for datasets with few SNPs, high proportions of missing data, or if the data is expected to be noisy. With the exemplary analysis of the empirical *HO-WE-Çayönü* dataset, we further demonstrated the utility of Pandora stability values for preventing potentially erroneous and hence misleading conclusions based on unstable dimensionality reduction results.

We believe that Pandora will be of high value to population genetic studies as the inherent uncertainty of PCA and MDS analyses can now be seamlessly and routinely assessed via an easy-to-use tool and thereby contribute to circumventing potentially biased conclusions.

In Pandora, the bootstrapping and the subsequent dimensionality reduction are both performed along the SNPs. Pandora does not implement a resampling of the individuals, since Elhaik [41] demonstrated the sensitivity of PCA studies to (subjective) selection of individuals. The author showed that PCA analyses can be manipulated to confirm the desired hypothesis, depending on the selection of individuals used for PCA analyses. Consequently, for Pandora, we refrained from sampling individuals. Thus, Pandora results cannot directly be used to justify a subjective selection of individuals. However, such an individual-sampling-based procedure might allow estimating the bias induced by specific individuals. In conjunction with the Pandora stability estimates, especially the PSVs, this can contribute to identifying an appropriate, meaningful set of individuals for population genetics studies. Devising such a selection criterion constitutes the subject of future work.

As previously mentioned, despite their popularity, the use of PCA or MDS in population genetics is controversial. More recent dimensionality reduction approaches such as t-SNE [179] or deep learning based variational autoencoders [13] alleviate some problems, but exhibit inherent disadvantages (see Battey *et al.* [13] for a discussion). Integrating such methods into Pandora as alternatives to PCA or MDS also constitutes future work.

6. Conclusion and Outlook

In this thesis, I outlined three novel applications of machine learning and data science methods to evolutionary genomics. I developed Pythia, a tool that predicts the difficulty of a phylogenetic inference prior to conducting time-consuming analyses, thus allowing evolutionary biologists to perform a more systematic and informed data analysis. Furthermore, I demonstrated a lack of realism in phylogenetic data simulations and identified the respective root causes. Finally, I presented Pandora, a tool for estimating uncertainty of dimensionality reduction techniques that are applied to population genetics data. Pandora seamlessly integrates uncertainty estimation and analysis via a single software tool.

Initially, I presented *Pythia*, a machine learning-based framework that predicts the difficulty of MSAs under ML phylogenetic analyses. I proposed a novel quantification of the "ruggedness" of the ML tree space and demonstrated that this quantification accurately captures the difficulty of an MSA. I trained a machine learning model that predicts this difficulty with high accuracy based on fast-to-compute prediction features. These features include MSA attributes, measures of information content, and features that approximate the properties of the ML tree space via MP trees. Pythia allows predicting the difficulty *before* conducting any time- and resource-intensive ML-based phylogenetic analyses. I showed that predicting the difficulty of an MSA using Pythia is substantially faster than conducting a single ML tree inference. Therefore, Pythia presents a major step towards faster and more systematic data analyses. Finally, the published applications of Pythia as outlined in Section 3.3.5 prove the high impact of Pythia and its utility, not only for phylogenetic analyses, but also for phylogenetics research.

I also presented *Pandora*, a software tool to estimate the stability of genotype data under dimensionality reduction. Pandora estimates the stability via bootstrapping and supports PCA and MDS analyses. Through three distinct stability criteria, Pandora allows exploring various data instability aspects. The PS value estimates the overall data stability, while the PSVs provide a per-individual stability estimate.

This is particularly useful if the projection of single individuals is of interest, for example, in ancient DNA studies. Finally, Pandora estimates the stability of clustering individuals into (sub-)populations via the PCS. Using empirical and simulated genotype data, I demonstrated that Pandora can detect instability in both, PCA, and MDS analyses. I further demonstrated the usability and utility of Pandora, and its potential to unravel potentially misleading results that are induced by unstable analyses. While the implemented bootstrapping procedure induces a substantial runtime overhead, I showed that my bootstrap convergence criterion can alleviate this overhead to some extent.

To ensure an easy usability, Pythia and Pandora are available as command line tools as well as Python libraries on GitHub. Both tools are well-tested, production ready, and include a thorough documentation. To simplify their installation, I also made both tools available on *conda-forge*, a popular open-source package repository.

Finally, in a collaborative study, Johanna Trost, Dimitri Höhler, and I demonstrated a lack of simulation realism in phylogenetics. We simulated collections of MSAs using increasingly complex models of DNA and AA sequence evolution. We deployed two distinct machine learning-based classification approaches (GBTs and CNNs) to distinguish between empirical and simulated MSAs. The GBTs rely on hand-crafted prediction features, and therefore yield interpretable results. In contrast, the CNNs categorize an MSA based on the site-wise composition of nucleotides/AAs, and classify data more accurately than the GBTs. Overall, our classifiers easily distinguished empirical from simulated MSAs across all evolutionary models, indicating a general lack of realism in sequence simulation. We found that a major cause for unrealistic simulations is that the rate of evolution across sites, as well as the site-wise composition, is more uniform in simulations than in empirical data. Our approach provides a framework for assessing the realism of simulations for future research on evolutionary models or simulation tools.

6.1 Future Work

In this final section of my thesis, I will outline possible directions of future work for all research projects I presented in this thesis.

6.1.1 Pythia

In our initial published version, we observed that Pythia tends to overestimate the difficulty of *easy* MSAs, and to underestimate the difficulty of *hard-to-analyze* MSAs. We hypothesize that this effect is caused by the abundance of *easy* MSAs in Pythia's training data. The training data for our latest Pythia version comprises approximately three times more MSAs than our initial dataset, and includes more intermediate and hard-to-analyze MSAs. As a result, the respective over- and underestimation is less pronounced. For a next version of the difficulty predictor, one could include more intermediate and hard-to-analyze MSAs in the training procedure to attain a more even distribution of difficulty levels.

Pythia's runtime is dominated by the MP-based feature computations. In our latest Pythia version, we were able to reduce the absolute runtime by reducing the number of inferred MP trees from 100 to 24. Yet, the MP-based features still account for approximately 70% of the runtime. Currently, Pythia relies on RAxML-NG for MP tree inference. Future work should therefore focus on either improving the MP tree inference runtime in RAxML-NG (which is developed in our research group), or by replacing RAxML-NG with a faster alternative.

Since the initial publication of Pythia in 2022, difficulty prediction has proven useful across various applications beyond a more informed analysis setup (see Section 3.3.5 for an overview). Exploring additional applications could be the subject of future work. For instance, expanding on our explorative analyses on the correlation between Pythia's difficulty prediction and MCMC convergence could be valuable to demonstrate the utility of difficulty prediction for tree inference methods beyond ML. Furthermore, Pythia difficulty scores can serve as a useful criterion for assembling collections of MSAs for testing and benchmarking novel inference tools. Such MSA collections should ideally comprise MSAs across all difficulty levels to ensure a representative experimental setup. Finally, one goal of this thesis was to explore novel tools for systematic data analysis. While Pythia currently predicts the difficulty of an assembled MSA, one could also devise a sequence selection criterion using Pythia to find the most promising (sub)set of sequences for a specific phylogenetic analysis. Given a set of aligned sequences (possibly comprising multiple sequences for the same species), one could search for a subset of these sequences that yields an easier MSA.

6.1.2 Simulations of Sequence Evolution

In our study on the realism of sequence simulations, we analyzed a plethora of DNA and AA evolutionary models. For DNA models, we focused on nucleotide substitution models. These models do not make any assumptions about the type of the underlying DNA. The DNA can be broadly categorized into coding and non-coding regions [185]. Only the coding regions of the DNA encode proteins, with three nucleotides forming a so-called *codon*, which corresponds to a single AA. Dedicated codon substitution models can take these restrictions into account. Thus, simulating MSAs of coding regions via codon models might yield more empirical-like data. Testing this hypothesis could be the subject of future studies.

Furthermore, with recent advances in deep learning, one could design and implement novel sequence simulation tools. Using approaches such as *Generative Adversarial Networks* [62] might allow for model-less simulations without imposing an artificial bias such as the assumption of uniform evolution along the MSA. Yet, designing and training such a network is a challenging task and requires careful consideration of all components. Especially, the integration of a ground-truth phylogenetic tree to simulate an MSA on is a major challenge.

6.1.3 Pandora

Pandora estimates the stability of genotype data via bootstrapping. This bootstrapping, and the subsequent dimensionality reduction, are performed along the

SNPs rather than along the individuals. This is because we assume statistical independence between the SNPs, whereas we cannot assume independence among the individuals within a single population (see Chapter 5). Yet, sampling individuals and subsequently estimating the data stability under dimensionality reduction might help to estimate the bias induced by single individuals. Furthermore, such an individual-based sampling and subsequent stability estimation might serve as a sample selection criterion that helps researchers to determine the optimal set of individuals to use in their analyses.

The implemented bootstrapping procedure for stability estimation is a time-consuming procedure, since Pandora repeatedly performs dimensionality reduction on the genotype data. Future work should focus on finding a shortcut to obtain a reliable stability estimate without this runtime overhead. Using, for instance, a machine learning model and carefully selected, hand-crafted prediction features could yield a fast approximation of genotype data stability under dimensionality reduction.

Currently, Pandora only supports stability estimation for PCA and MDS analyses. While PCA and MDS are still being widely used, more recent approaches such as t-SNE or variational autoencoders are becoming increasingly popular. To provide a universal solution for stability estimation of dimensionality reduction in population genetics research, one could include such alternative techniques in Pandora.

Bibliography

[1] S. Abadi, O. Avram, S. Rosset, T. Pupko and I. Mayrose. ModelTeller: model selection for optimal phylogenetic reconstruction using machine learning. *Molecular Biology and Evolution*, 37(11):3338–3352, 2020.

[2] H. Abdi. Bonferroni and Šidák corrections for multiple comparisons. *Encyclopedia of Measurement and Statistics*, 3(01):2007, 2007.

[3] J. R. Adrion, C. B. Cole, N. Dukler, J. G. Galloway, A. L. Gladstein, G. Gower, C. C. Kyriazis, A. P. Ragsdale, G. Tsambos *et al.* A community-maintained standard library of population genetic models. *eLife*, 9, 2020.

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama. Optuna. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, New York, NY, USA, 2019. ACM.

[5] B. Alipanahi, A. Delong, M. T. Weirauch and B. J. Frey. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, 2015.

[6] D. Aloise, A. Deshpande, P. Hansen and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[7] N. E. Altınışık, D. D. Kazancı, A. Aydoğan, H. C. Gemici, Ö. D. Erdal, S. Sarıaltun, K. B. Vural, D. Koptekin, K. Gürün *et al.* A genomic snapshot of demographic and cultural dynamism in Upper Mesopotamia during the Neolithic Transition. *Science Advances*, 8(44), 2022.

[8] E. Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457, 1936.

[9] C. Angermueller, T. Pärnamaa, L. Parts and O. Stegle. Deep learning for computational biology. *Molecular Systems Biology*, 12(7):878, 2016.

[10] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.

[11] Q. D. Atkinson and R. D. Gray. Curious parallels and curious connections – phylogenetic thinking in biology and historical linguistics. *Systematic Biology*, 54(4):513–526, 2005.

[12] H.-J. Bandelt and A. W. M. Dress. Split decomposition: a new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*, 1(3):242–252, 1992.

[13] C. J. Battey, G. C. Coffing and A. D. Kern. Visualizing population structure with variational autoencoders. *G3 Genes|Genomes|Genetics*, 11(1):jkaa036, 2021.

[14] F. Baumdicker, G. Bisschop, D. Goldstein, G. Gower, A. P. Ragsdale, G. Tsambos, S. Zhu, B. Eldon, E. C. Ellerman *et al.* Efficient ancestry and mutation simulation with msprime 1.0. *Genetics*, 220(3), 2022.

[15] J. Bergsten. A review of long-branch attraction. *Cladistics*, 21(2):163–193, 2005.

[16] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[17] M. Bernabeu, S. Manzano-Morales, M. Marcet-Houben and T. Gabaldón. Diverse ancestries reveal complex symbiotic interactions during eukaryogenesis. *bioRxiv*, 2024.

[18] B. Bettisworth and A. Stamatakis. Root Digger: a root placement program for phylogenetic trees. *BMC Bioinformatics*, 22(1):225, 2021.

[19] M. K. Bhattacharyya, A. M. Smith, T. Ellis, C. Hedley and C. Martin. The wrinkled-seed character of pea described by Mendel is caused by a transposon-like insertion in a gene encoding starch-branching enzyme. *Cell*, 60(1):115–122, 1990.

[20] J. P. Bollback. Bayesian model adequacy and choice in phylogenetics. *Molecular Biology and Evolution*, 19(7):1171–1180, 2002.

[21] B. E. Boser, I. M. Guyon and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

[22] R. Bouckaert, T. G. Vaughan, J. Barido-Sottani, S. Duchêne, M. Fourment, A. Gavryushkina, J. Heled, G. Jones, D. Kühnert *et al.* BEAST 2.5: An advanced software platform for Bayesian evolutionary analysis. *PLOS Computational Biology*, 15(4):e1006650, 2019.

[23] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[24] R. P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, 14(4):422–425, 1971.

[25] R. Bricout, D. Weil, D. Stroebel, A. Genovesio and H. Roest Crollius. Evolution is not uniform along coding sequences. *Molecular Biology and Evolution*, 40(3):2022–2024, 2023.

[26] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry *et al.* Language models are few-shot learners. *arXiv*, 2020.

[27] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[28] R. A. Cartwright. DNA assembly with gaps (Dawg): simulating sequence evolution. *Bioinformatics*, 21(Suppl_3):iii31–iii38, 2005.

[29] C. C. Chang, C. C. Chow, L. C. Tellier, S. Vattikuti, S. M. Purcell and J. J. Lee. Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigascience*, 4(1), 2015.

[30] J. Charamis, S. Balaska, P. Ioannidis, V. Dvořák, K. Mavridis, M. A. McDowell, P. Pavlidis, R. Feyereisen, P. Volf and J. Vontas. Comparative genomics uncovers the evolutionary dynamics of detoxification and insecticide target genes across 11 Phlebotomine sand flies. *Genome Biology and Evolution*, 16 (9), 2024.

[31] B. Chor and T. Tuller. Maximum likelihood of evolutionary trees is hard. In S. Miyano, J. Mesirov, S. Kasif, S. Istrail, P. A. Pevzner and M. Waterman, editors, *Research in Computational Molecular Biology*, pages 296–310, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[32] L. H. Clemmensen and R. D. Kjærsgaard. Data representativity for machine learning and AI systems. *arXiv*, 2022.

[33] L. Collienne, M. Barker, M. A. Suchard and F. A. Matsen. Phylogenetic tree instability after taxon addition: empirical frequency, predictability, and consequences for online inference. *Systematic Biology*, 74(1):101–111, 2025.

[34] K. Csilléry, M. G. B. Blum, O. E. Gaggiotti and O. François. Approximate Bayesian Computation (ABC) in practice. *Trends in Ecology & Evolution*, 25 (7):410–418, 2010.

[35] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1859.

[36] W. H. E. Day. Analysis of quartet dissimilarity measures between undirected phylogenetic trees. *Systematic Biology*, 35(3):325–333, 1986.

[37] N. Ecker, D. Huchon, Y. Mansour, I. Mayrose and T. Pupko. A machine-learning-based alternative to phylogenetic bootstrap. *Bioinformatics*, 40 (Supplement_1):i208–i217, 2024.

[38] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[39] R. C. Edgar. Muscle5: high-accuracy alignment ensembles enable unbiased assessments of sequence homology and phylogeny. *Nature Communications*, 13(1):6968, 2022.

[40] B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7(1), 1979.

[41] E. Elhaik. Principal component analyses (PCA)-based findings in population genetic studies are highly biased and must be reevaluated. *Scientific Reports*, 12(1):14683, 2022.

[42] G. F. Estabrook, F. R. McMorris and C. A. Meacham. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Systematic Zoology*, 34(2):193–200, 1985.

[43] Exelixis-Lab. COre RAXml LIBrary (Coraxlib), 2022. Online: `https://codeberg.org/Exelixis-Lab/coraxlib`.

[44] J. S. Farris. Methods for computing Wagner trees. *Systematic Biology*, 19(1): 83–92, 1970.

[45] S. Faulwetter, A. Vasileiadou, M. Kouratoras, T. Dailianis and C. Arvanitidis. Micro-computed tomography: introducing new dimensions to taxonomy. *ZooKeys*, 263:1–45, 2013.

[46] J. Felsenstein. Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters. *Systematic Zoology*, 22(3):240–249, 1973.

[47] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[48] J. Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4):783–791, 1985.

[49] J. Felsenstein. *Inferring Phylogenies*. Sinauer, 2003.

[50] A. Fisher, B. Caffo, B. Schwartz and V. Zipunnikov. Fast, exact bootstrap principal component analysis for p $>$ 1 million. *Journal of the American Statistical Association*, 111(514):846–860, 2016.

[51] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[52] W. M. Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Zoology*, 20(4):406–416, 1971.

[53] R. Fletcher. Newton-like methods. In *Practical Methods of Optimization*, chapter 3, pages 44–79. John Wiley & Sons, Ltd, 2000.

[54] W. Fletcher and Z. Yang. INDELible: a flexible simulator of biological sequence evolution. *Molecular Biology and Evolution*, 26(8):1879–1888, 2009.

[55] L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3(1):43–49, 1982.

[56] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553, 1983.

[57] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pages 148–156, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[58] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[59] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5), 2001.

[60] N. Goldman, J. P. Anderson and A. G. Rodrigo. Likelihood-based tests of topologies in phylogenetics. *Systematic Biology*, 49(4):652–670, 2000.

[61] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. MIT Press, 2016.

[62] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[63] G. W. Grimm, S. S. Renner, A. Stamatakis and V. Hemleben. A nuclear ribosomal dna phylogeny of Acer inferred with maximum likelihood, splits graphs, and motif analysis of 606 sequences. *Evolutionary Bioinformatics*, 2: 7–22, 2006.

[64] S. Guindon, J.-F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk and O. Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. *Systematic Biology*, 59(3): 307–321, 2010.

[65] J. Haag and A. Stamatakis. Pythia 2.0: new data, new prediction model, new features. *bioRxiv*, 2025.

[66] J. Haag, D. Höhler, B. Bettisworth and A. Stamatakis. From easy to hopeless – predicting the difficulty of phylogenetic analyses. *Molecular Biology and Evolution*, 39(12):msac254, 2022.

[67] J. Haag, L. Hübner, A. M. Kozlov and A. Stamatakis. The Free Lunch is not over yet – systematic exploration of numerical thresholds in maximum likelihood phylogenetic inference. *Bioinformatics Advances*, 3(1), 2023.

[68] J. Haag, A. I. Jordan and A. Stamatakis. Pandora: a tool to estimate dimensionality reduction stability of genotype data. *Bioinformatics Advances*, 5(1), 2024.

[69] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg *et al.* Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[70] M. Hasegawa, H. Kishino and T.-a. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22(2):160–174, 1985.

[71] T. Hastie, R. Tibshirani and J. Friedman. Boosting and additive trees. In T. Hastie, R. Tibshirani and J. Friedman, editors, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, pages 337–387. Springer New York, New York, NY, 2009.

[72] T. Hastie, R. Tibshirani and J. Friedman. Model assessment and selection. In T. Hastie, R. Tibshirani and J. Friedman, editors, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, pages 219–259. Springer New York, New York, NY, 2009.

[73] T. Hastie, R. Tibshirani and J. Friedman. Unsupervised learning. In T. Hastie, R. Tibshirani and J. Friedman, editors, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, pages 485–585. Springer New York, New York, NY, 2009.

[74] S. B. Hedges, K. D. Moberg and L. R. Maxson. Tetrapod phylogeny inferred from 18S and 28S ribosomal RNA sequences and a review of the evidence for amniote relationships. *Molecular Biology and Evolution*, 7(6):607–633, 1990.

[75] T. K. Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282, 1995.

[76] E. B. Hodcroft, N. De Maio, R. Lanfear, D. R. MacCannell, B. Q. Minh, H. A. Schmidt, A. Stamatakis, N. Goldman and C. Dessimoz. Want to track pandemic variants faster? Fix the bioinformatics bottleneck. *Nature*, 591 (7848):30–33, 2021.

[77] D. Hoehler, J. Haag, A. M. Kozlov and A. Stamatakis. A representative performance assessment of maximum likelihood based phylogenetic inference tools. *bioRxiv*, 2022.

[78] M. Höhl and M. A. Ragan. Is multiple-sequence alignment required for accurate inference of phylogeny? *Systematic Biology*, 56(2):206–221, 2007.

[79] D. Höhler, W. Pfeiffer, V. Ioannidis, H. Stockinger and A. Stamatakis. RAxML Grove: an empirical phylogenetic tree database. *Bioinformatics*, 38(6):1741–1742, 2021.

[80] S. Höhna, M. J. Landis, T. A. Heath, B. Boussau, N. Lartillot, B. R. Moore, J. P. Huelsenbeck and F. Ronquist. RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Systematic Biology*, 65(4):726–736, 2016.

[81] B. R. Holland, K. T. Huber, A. Dress and V. Moulton. δ Plots: a tool for analyzing phylogenetic distance data. *Molecular Biology and Evolution*, 19 (12):2051–2059, 2002.

[82] T. Hu, N. Chitnis, D. Monos and A. Dinh. Next-generation sequencing technologies: an overview. *Human Immunology*, 82(11):801–811, 2021.

[83] J. R. Hughey, P. Paschou, P. Drineas, D. Mastropaolo, D. M. Lotakis, P. A. Navas, M. Michalodimitrakis, J. A. Stamatoyannopoulos and G. Stamatoyannopoulos. A European population in Minoan Bronze Age Crete. *Nature Communications*, 4(1):1861, 2013.

[84] G. James, D. Witten, T. Hastie and R. Tibshirani. *An Introduction to Statistical Learning*. Springer Texts in Statistics. Springer US, New York, NY, 2 edition, 2021.

[85] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In H. N. MUNRO, editor, *Mammalian Protein Metabolism*, pages 21–132. Elsevier, 1969.

[86] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[87] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615–623, 2001.

[88] K. Katoh, K. Misawa, K. Kuma and T. Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14):3059–3066, 2002.

[89] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T.-Y. Liu. LightGBM: a highly efficient gradient boosting decision tree. In *Proceedings of*

the 31st International Conference on Neural Information Processing Systems, NIPS'17, pages 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc.

[90] J. Kingman. The coalescent. *Stochastic Processes and their Applications*, 13 (3):235–248, 1982.

[91] H. Kishino and M. Hasegawa. Evaluation of the maximum likelihood estimate of the evolutionary tree topologies from DNA sequence data, and the branching order in hominoidea. *Journal of Molecular Evolution*, 29(2):170–179, 1989.

[92] H. Kishino, T. Miyata and M. Hasegawa. Maximum likelihood inference of protein phylogeny and the origin of chloroplasts. *Journal of Molecular Evolution*, 31(2):151–160, 1990.

[93] D. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Professional, 1997.

[94] J. Köster and S. Rahmann. Snakemake – a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[95] P. Kotsakiozi, A. Antoniou, N. Psonis, K. Sagonas, E. Karameta, Ç. Ilgaz, Y. Kumlutaş, A. Avcı, D. Jablonski *et al.* Cryptic diversity and phylogeographic patterns of Mediodactylus species in the Eastern Mediterranean region. *Molecular Phylogenetics and Evolution*, 197:108091, 2024.

[96] A. M. Kozlov, D. Darriba, T. Flouri, B. Morel and A. Stamatakis. RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 35(21):4453–4455, 2019.

[97] A. Kryshtafovych, T. Schwede, M. Topf, K. Fidelis and J. Moult. Critical assessment of methods of protein structure prediction (CASP) – Round XIV. *Proteins: Structure, Function, and Bioinformatics*, 89(12):1607–1617, 2021.

[98] M. K. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11(3):459–468, 1994.

[99] S. Kumar, Q. Tao, A. P. Lamarca and K. Tamura. Computational reproducibility of molecular phylogenies. *Molecular Biology and Evolution*, 40(7), 2023.

[100] C. Lakner, P. van der Mark, J. P. Huelsenbeck, B. Larget and F. Ronquist. Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics. *Systematic Biology*, 57(1):86–103, 2008.

[101] M. E. Lauterbur, M. I. A. Cavassim, A. L. Gladstein, G. Gower, N. S. Pope, G. Tsambos, J. Adrion, S. Belsare, A. Biddanda *et al.* Expanding the stdpopsim species catalog, and lessons learned for realistic genome simulations. *eLife*, 12, 2023.

[102] I. Lazaridis, N. Patterson, A. Mittnik, G. Renaud, S. Mallick, K. Kirsanow, P. H. Sudmant, J. G. Schraiber, S. Castellano *et al.* Ancient human genomes suggest three ancestral populations for present-day Europeans. *Nature*, 513 (7518):409–413, 2014.

[103] S. Q. Le and O. Gascuel. An improved general amino acid replacement matrix. *Molecular Biology and Evolution*, 25(7):1307–1320, 2008.

[104] S. Q. Le, O. Gascuel and N. Lartillot. Empirical profile mixture models for phylogenetic reconstruction. *Bioinformatics*, 24(20):2317–2323, 2008.

[105] S. Q. Le, N. Lartillot and O. Gascuel. Phylogenetic mixture models for proteins. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 363(1512):3965–3976, 2008.

[106] Y. LeCun, Y. Bengio and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[107] P. Linardatos, V. Papastefanopoulos and S. Kotsiantis. Explainable AI: a review of machine learning interpretability methods. *Entropy*, 23(1):18, 2020.

[108] C. Liu, X. Zhou, Y. Li, C. T. Hittinger, R. Pan, J. Huang, X.-x. Chen, A. Rokas, Y. Chen and X.-X. Shen. The influence of the number of tree searches on maximum likelihood inference in phylogenomics. *Systematic Biology*, 73(5):807–822, 2024.

[109] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[110] G. Loewenthal, D. Rapoport, O. Avram, A. Moshe, E. Wygoda, A. Itzkovitch, O. Israeli, D. Azouri, R. A. Cartwright *et al.* A probabilistic model for indel evolution: differentiating insertions from deletions. *Molecular Biology and Evolution*, 38(12):5769–5781, 2021.

[111] J.-M. Lueckmann, J. Boelts, D. Greenberg, P. Goncalves and J. Macke. Benchmarking simulation-based inference. In A. Banerjee and K. Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 343–351. PMLR, 2021.

[112] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[113] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal and S.-I. Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, 2020.

[114] N. Ly-Trong, S. Naser-Khdour, R. Lanfear and B. Q. Minh. AliSim: a fast and versatile phylogenetic sequence simulator for the genomic era. *Molecular Biology and Evolution*, 39(5), 2022.

[115] J. Lyons-Weiler, G. A. Hoelzer and R. J. Tausch. Relative apparent synapomorphy analysis (RASA). I: the statistical measurement of phylogenetic signal. *Molecular Biology and Evolution*, 13(6):749–757, 1996.

[116] A. Mead. Review of the development of multidimensional scaling methods. *The Statistician*, 41(1):27, 1992.

[117] G. Mendel. Versuche über Pflanzen-Hybriden. *Verhandlungen des Naturforschenden Vereines in Brünn*, 4:3–47, 1866.

[118] P. Menozzi, A. Piazza and L. Cavalli-Sforza. Synthetic maps of human gene frequencies in Europeans. *Science*, 201(4358):786–792, 1978.

[119] M. L. Metzker, D. P. Mindell, X.-M. Liu, R. G. Ptak, R. A. Gibbs and D. M. Hillis. Molecular evidence of HIV-1 transmission in a criminal case. *Proceedings of the National Academy of Sciences*, 99(22):14292–14297, 2002.

[120] A. Miles, p. bot, M. R., P. Ralph, J. Kelleher, M. Schelker, R. Pisupati, S. Rae and T. Millar. cggh/scikit-allel: v1.3.7, 2023. Online: `https://doi.org/10.5281/zenodo.8326460`.

[121] B. Q. Minh, H. A. Schmidt, O. Chernomor, D. Schrempf, M. D. Woodhams, A. von Haeseler and R. Lanfear. IQ-TREE 2: new models and efficient methods for phylogenetic inference in the genomic era. *Molecular Biology and Evolution*, 37(5):1530–1534, 2020.

[122] A. A. Miranda, Y.-A. Le Borgne and G. Bontempi. New routes from minimal approximation error to principal components. *Neural Processing Letters*, 27 (3):197–207, 2008.

[123] B. Misof, K. Meusemann, B. M. von Reumont, P. Kück, S. J. Prohaska and P. F. Stadler. A priori assessment of data quality in molecular phylogenetics. *Algorithms for Molecular Biology*, 9(1):22, 2014.

[124] J. Mockus, V. Tiesis and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

[125] B. Morel, P. Barbera, L. Czech, B. Bettisworth, L. Hübner, S. Lutteropp, D. Serdari, E.-G. Kostaki, I. Mamais *et al.* Phylogenetic analysis of SARS-CoV-2 data is difficult. *Molecular Biology and Evolution*, 38(5):1777–1791, 2020.

[126] K. Morrill, J. Hekman, X. Li, J. McClure, B. Logan, L. Goodman, M. Gao, Y. Dong, M. Alonso *et al.* Ancestry-inclusive dog genomics challenges popular breed stereotypes. *Science*, 376(6592), 2022.

[127] National Center for Biotechnology Information. GenBank and WGS statistics, 2025. Online: https://www.ncbi.nlm.nih.gov/genbank/statistics/.

[128] National Human Genome Research Institute. The cost of sequencing a human genome, 2021. Online: https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost.

[129] L. Nesterenko, L. Blassel, P. Veber, B. Boussau and L. Jacob. Phyloformer: fast, accurate, and versatile phylogenetic reconstruction with deep neural networks. *Molecular Biology and Evolution*, 42(4), 2025.

[130] T. H. Ogden and M. S. Rosenberg. Multiple sequence alignment accuracy and phylogenetic inference. *Systematic Biology*, 55(2):314–328, 2006.

[131] G. Papamakarios and I. Murray. Fast ϵ-free inference of simulation models with Bayesian conditional density estimation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1036–1044. Curran Associates Inc., 2016.

[132] J. Parker, G. Tsagkogeorga, J. A. Cotton, Y. Liu, P. Provero, E. Stupka and S. J. Rossiter. Genome-wide signatures of convergent evolution in echolocating mammals. *Nature*, 502(7470):228–231, 2013.

[133] N. Patterson, A. L. Price and D. Reich. Population structure and eigenanalysis. *PLoS Genetics*, 2(12):e190, 2006.

[134] K. Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[135] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss *et al.* Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[136] S. Penel, A.-M. Arigon, J.-F. Dufayard, A.-S. Sertier, V. Daubin, L. Duret, M. Gouy and G. Perrière. Databases of homologous gene families for comparative genomics. *BMC Bioinformatics*, 10(6):S3, 2009.

[137] W. Piel, M. Donoghue, M. Sanderson and C. Person. TreeBASE: a database of phylogenetic information. 2000.

[138] W. H. Piel, L. Chan, M. J. Dominus, J. Ruan, R. A. Vos and V. Tannen. TreeBASE v.2: a database of phylogenetic knowledge. *e-BioSphere 2009*, 2009.

[139] N. Poulakakis and A. Stamatakis. Recapitulating the evolution of Afrotheria: 57 genes and rare genomic changes (RGCs) consolidate their history. *Systematics and Biodiversity*, 8(3):395–408, 2010.

[140] A. L. Price, N. J. Patterson, R. M. Plenge, M. E. Weinblatt, N. A. Shadick and D. Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics*, 38(8):904–909, 2006.

[141] M. N. Price, P. S. Dehal and A. P. Arkin. FastTree: computing large minimum evolution trees with profiles instead of a distance matrix. *Molecular Biology and Evolution*, 26(7):1641–1650, 2009.

[142] M. N. Price, P. S. Dehal and A. P. Arkin. FastTree 2 – approximately maximum-likelihood trees for large alignments. *PLOS ONE*, 5(3):1–10, 2010.

[143] S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. de Bakker *et al.* PLINK: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007.

[144] H. Ren, T. K. F. Wong, B. Q. Minh and R. Lanfear. Mixturefinder: estimating DNA mixture models for phylogenetic analyses. *Molecular Biology and Evolution*, 42(1), 2025.

[145] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981.

[146] F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard and J. P. Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology*, 61(3):539–542, 2012.

[147] M. S. Rosenberg and S. Kumar. Incomplete taxon sampling is not a problem for phylogenetic inference. *Proceedings of the National Academy of Sciences*, 98(19):10751–10756, 2001.

[148] N. A. Rosenberg, J. K. Pritchard, J. L. Weber, H. M. Cann, K. K. Kidd, L. A. Zhivotovsky and M. W. Feldman. Genetic structure of human populations. *Science*, 298(5602):2381–2385, 2002.

[149] D. E. Rumelhart, G. E. Hinton and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[150] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[151] H. A. Schmidt. Testing tree topologies. In *The Phylogenetic Handbook*, The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing, pages 381–404. Cambridge University Press, Cambridge, 2 edition, 2009.

[152] R. Schneider and C. Sander. The HSSP database of protein structure-sequence alignments. *Nucleic Acids Research*, 24(1):201–205, 1996.

[153] D. Schrempf, N. Lartillot and G. Szöllősi. Scalable empirical mixture models that account for across-site compositional heterogeneity. *Molecular Biology and Evolution*, 37(12):3616–3631, 2020.

[154] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2), 1978.

[155] D. W. Scott. *Multivariate Density Estimation*. Wiley Series in Probability and Statistics. Wiley, 1992.

[156] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[157] L. S. Shapley. 17. A Value for n-person games. In *Contributions to the Theory of Games (AM-28), Volume II*, pages 307–318. Princeton University Press, 1953.

[158] H. Shimodaira. An approximately unbiased test of phylogenetic tree selection. *Systematic Biology*, 51(3):492–508, 2002.

[159] H. Shimodaira and M. Hasegawa. Multiple comparisons of log-likelihoods with applications to phylogenetic inference. *Molecular Biology and Evolution*, 16 (8):1114, 1999.

[160] J. Shlens. A tutorial on principal component analysis. *arXiv*, 2014.

[161] J. S. Shoemaker and W. M. Fitch. Evidence from nuclear sequences that invariable sites should be considered when sequence divergence is calculated. *Molecular Biology and Evolution*, 6(3):270–289, 1989.

[162] L. Si Quang, O. Gascuel and N. Lartillot. Empirical profile mixture models for phylogenetic reconstruction. *Bioinformatics*, 24(20):2317–2323, 2008.

[163] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert *et al.* Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(1):539, 2011.

[164] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas science bulletin*, 38:1409–1438, 1958.

[165] N. Solovieff, S. W. Hartley, C. T. Baldwin, T. T. Perls, M. H. Steinberg and P. Sebastiani. Clustering by genetic ancestry using genome-wide SNP data. *BMC Genetics*, 11(1):108, 2010.

[166] A. Stamatakis. Phylogenetic search algorithms for maximum likelihood. In *Algorithms in Computational Molecular Biology*, chapter 25, pages 547–577. John Wiley & Sons, Ltd, 2011.

[167] A. Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

[168] A. Stamatakis and A. M. Kozlov. Efficient maximum likelihood tree building methods. In C. Scornavacca, F. Delsuc and N. Galtier, editors, *Phylogenetics in the Genomic Era*, page 1.2:1–1.2:18. No commercial publisher | Authors open access book, 2020.

[169] A. Stamatakis, T. Ludwig and H. Meier. RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, 2004.

[170] J. Stiller, S. Feng, A.-A. Chowdhury, I. Rivas-González, D. A. Duchêne, Q. Fang, Y. Deng, A. Kozlov, A. Stamatakis *et al.* Complexity of avian evolution revealed by family-level genomes. *Nature*, 629(8013):851–860, 2024.

[171] K. Strimmer and A. Rambaut. Inferring confidence sets of possibly misspecified gene trees. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 269(1487):137–142, 2002.

[172] A. Suvorov, J. Hochuli and D. R. Schrider. Accurate inference of tree topologies from multiple sequence alignments using deep learning. *Systematic Biology*, 69(2):221–233, 2019.

[173] S. Tavaré. Some probabilistic and statistical problems on the analysis of DNA sequences. *Lectures on Mathematics in the Life Sciences*, 17:57–86, 1986.

[174] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[175] A. Togkousidis, O. M. Kozlov, J. Haag, D. Höhler and A. Stamatakis. Adaptive RAxML-NG: accelerating phylogenetic inference under maximum likelihood using dataset difficulty. *Molecular Biology and Evolution*, 40(10), 2023.

[176] A. Togkousidis, A. Stamatakis and O. Gascuel. Much ado about nothing: accelerating maximum likelihood phylogenetic inference via early stopping to evade (over-)optimization. *bioRxiv*, 2024.

[177] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[178] J. Trost, J. Haag, D. Höhler, L. Jacob, A. Stamatakis and B. Boussau. Simulations of sequence evolution: how (un)realistic they are and why. *Molecular Biology and Evolution*, 41(1), 2024.

[179] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[180] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[181] L. S. Vinh and A. von Haeseler. IQPNNI: moving fast through tree space and stopping in time. *Molecular Biology and Evolution*, 21(8):1565–1571, 2004.

[182] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

[183] R. A. Vos, J. P. Balhoff, J. A. Caravas, M. T. Holder, H. Lapp, W. P. Maddison, P. E. Midford, A. Priyam, J. Sukumaran *et al.* NeXML: rich, extensible, and verifiable representation of comparative data and metadata. *Systematic Biology*, 61(4):675–689, 2012.

[184] C. Wang, Z. A. Szpiech, J. H. Degnan, M. Jakobsson, T. J. Pemberton, J. A. Hardy, A. B. Singleton and N. A. Rosenberg. Comparing spatial maps of human population-genetic variation using Procrustes analysis. *Statistical Applications in Genetics and Molecular Biology*, 9(1), 2010.

[185] J. D. Watson, T. A. Baker, S. P. Bell, A. Gann, M. Levine and R. Losick. *Watson Molekularbiologie*. Pearson Deutschland, 2010.

[186] S. Whelan and N. Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular Biology and Evolution*, 18(5):691–699, 2001.

[187] W. T. White, S. F. Hills, R. Gaddam, B. R. Holland and D. Penny. Treeness triangles: visualizing the loss of phylogenetic signal. *Molecular Biology and Evolution*, 24(9):2029–2039, 2007.

[188] J. Wiegert, D. Höhler, J. Haag and A. Stamatakis. Predicting phylogenetic bootstrap values via machine learning. *Molecular Biology and Evolution*, 41 (10), 2024.

[189] T. A. Williams, C. J. Cox, P. G. Foster, G. J. Szöllősi and T. M. Embley. Phylogenomics provides robust support for a two-domains tree of life. *Nature Ecology & Evolution*, 4(1):138–147, 2019.

[190] S. Wright. The genetical structure of populations. *Annals of Eugenics*, 15(1): 323–354, 1949.

[191] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: theory and practice. *Neurocomputing*, 415:295–316, 2020.

[192] Z. Yang. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution*, 39(3):306–314, 1994.

[193] Z. Yang. Among-site rate variation and its impact on phylogenetic analyses. *Trends in Ecology & Evolution*, 11(9):367–372, 1996.

[194] Z. Yang. *Computational Molecular Evolution*. Oxford Series in Ecology and Evolution. Oxford University PressOxford, 2006.

[195] Z. Yang, N. Goldman and A. Friday. Maximum likelihood trees from DNA sequences: a peculiar statistical estimation problem. *Systematic Biology*, 44: 384–399, 1995.

[196] X. Yi and E. K. Latch. Nonrandom missing data can bias principal component analysis inference of population genetic structure. *Molecular Ecology Resources*, 22(2):602–611, 2022.

[197] X. Yuan, D. J. Miller, J. Zhang, D. Herrington and Y. Wang. An overview of population genetic data simulation. *Journal of Computational Biology*, 19(1): 42–54, 2012.

[198] J. Zhou and O. G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12(10):931–934, 2015.

In accordance with the statement of the Deutsche Forschungsgemeinschaft[2] on September 21st 2023 regarding the usage of generative models for text and image generation, we want to indicate that generative models were used for the following purposes: Part of the code accompanying this thesis was developed using GitHub Copilot (based on GPT-4o) and Claude Sonnet 3.7 for assisting writing a small subset of inline code documentation (Doxygen) and auto-completion at subline and line level. All generated documentation and code auto-completions have been verified manually by the authors. Further, no code generation or other generative model has been used for algorithm design, study design, or data analyses. We used LanguageTool for grammar checking of all publications mentioned in this thesis, as well as all text in this thesis. All proposed edits have been manually evaluated and accepted by the authors. No image in this thesis was generated using generative models.

[2] https://www.dfg.de/de/aktuelles/neuigkeiten-themen/info-wissenschaft/2023/info-wissenschaft-23-72