Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Simen Kristian Lunde Wold

Genomic Prediction in Wild Populations: Gradient Boosting with Reduced SNP Sets and Feature Compression

Master's thesis in Applied Physics and Mathematics
Supervisor: Stefanie Muff
June 2025

NTNU
Norwegian University of Science and Technology

Simen Kristian Lunde Wold

Genomic Prediction in Wild Populations: Gradient Boosting with Reduced SNP Sets and Feature Compression

Master's thesis in Applied Physics and Mathematics
Supervisor: Stefanie Muff
June 2025

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Genomic prediction has long been utilized by biologists and breeders to enhance artificial selection in animal and plant breeding. Recently, its application has expanded into ecological and population genetics, focusing more on wild populations. Here, genomic prediction helps study the genetic architecture of traits and the assessment of adaptive potential, informing conservation strategies. Compared to in livestock, genomic prediction in wild populations faces greater challenges due to increased phenotypic variation driven by environmental rather than genetic factors. Traditional methods, such as linear animal models, leverage estimated relatedness among individuals to predict genetic values but are limited by their inability to capture non-linear effects.

In this master's thesis, we investigate whether modern machine learning techniques and simpler linear models can compete with the established animal model in genomic prediction. Using data from a wild population of house sparrows – including both genetic markers (SNPs) and environmental variables – we evaluate how well gradient boosted models, capable of capturing non-linear patterns, and ridge regression perform across different traits against the animal model. In addition, we explore the impact of subset selection and feature extraction on predictive accuracy.

Our analysis began with visualization of the genetic structure using t-distributed stochastic neighbor embedding (t-SNE), which revealed clear clustering among individuals. We then systematically varied the number of SNPs used to train the models and found that predictive accuracy for certain traits peaks around 60,000 SNPs for the tree-based models – suggesting an optimal balance in the bias-variance tradeoff. Based on that, we fixed the SNP input size at 64,000 for a comparative evaluation of three gradient boosting frameworks – XGBoost, LightGBM, and CatBoost – benchmarked against ridge regression and the animal model. Finally, we assessed whether compressed SNP features from PCA and autoencoders could offer any advantages when used as model input.

The results show that no single method performs best across all traits. While gradient boosted models are competitive for some phenotypes, the animal model remains the most consistently accurate. Ridge regression performs nearly on par with the animal model and outperforms the more complex boosting methods – highlighting it as a simple and efficient alternative. Feature compression offered no accuracy gains; however, our findings underscore the importance of model selection tailored to trait architecture – providing practical guidance for genomic prediction in wild populations.

Sammendrag

Genomisk prediksjon har lenge blitt brukt av biologer og avlere for å forbedre kunstig seleksjon i husdyr- og planteavl. I nyere tid har bruken også utvidet seg til økologi og populasjonsgenetikk, med økt fokus på ville populasjoner. Her benyttes genomisk prediksjon til å studere genetisk arkitektur og vurdere adaptivt potensial, noe som er relevant for bevaringsstrategier. Sammenlignet med i husdyrpopulasjoner er genomisk prediksjon i ville populasjoner mer utfordrende, ettersom fenotypisk variasjon i større grad skyldes miljømessige faktorer snarere enn genetiske forskjeller. Tradisjonelle metoder, som lineære dyremodeller, benytter estimert slektskap mellom individer for å predikere genetiske verdier, men er begrenset av at de ikke fanger opp ikke-lineære effekter.

I denne masteroppgaven undersøker vi om moderne maskinlæringsmetoder og enklere lineære modeller kan konkurrere med den etablerte dyremodellen i genomisk prediksjon. Ved bruk av data fra en vill populasjon av gråspurv – som inkluderer genetiske markører (SNP-er) og miljøvariabler – evaluerer vi hvor godt gradient boosting-modeller, som kan fange opp ikke-lineære mønstre, og ridge-regresjon presterer sammenlignet med dyremodellen. I tillegg undersøker vi hvordan subsett-seleksjon og faeture extraction-metoder påvirker prediksjonsevne.

Analysen startet med visualisering av genetisk struktur ved hjelp av t-distributed stochastic neighbor embedding (t-SNE), som avdekket tydelig klustering blant individene. Deretter varierte vi systematisk antallet SNP-er brukt i modellene, og fant at prediksjonsevnen for enkelte fenotyper nådde en topp rundt 60 000 SNP-er for trebaserte modeller – som indikerer en optimal bias-varians-balanse. Basert på dette valgte vi å bruke 64 000 SNP-er som input i en sammenligning av tre gradient boosting-rammeverk – XGBoost, LightGBM og CatBoost – opp mot ridge-regresjon og dyremodellen. Til slutt undersøkte vi om komprimerte SNP-representasjoner fra PCA og autoenkodere kunne forbedre modellene.

Resultatene viser at ingen enkeltmetode presterer best for alle egenskaper. Mens gradient boosting-modeller er konkurransedyktige for noen fenotyper, forblir dyremodellen den mest konsistente og treffsikre. Ridge-regresjon presterer nesten like godt som dyremodellen og overgår de mer komplekse boosting-metodene – som gjør den til et enkelt og effektivt alternativ. Komprimering av variabler ga ingen forbedring i prediksjonsevne, men resultatene våre fremhever viktigheten av å tilpasse modellvalg til fenotypenes genetiske arkitektur – og gir praktisk veiledning for genomisk prediksjon i ville populasjoner.

Table of Contents

List of Figures

List of Tables

1 Introduction

Dating back to the 1860s, Gregor Mendel's experiments on pea plants marked the birth of modern genetics (Griffiths et al., 2004). Mendel focused on traits with clear-cut, qualitative differences (*e.g.*, yellow versus green seeds), demonstrating that inherited factors (now known as genes) influence observable traits. However, most naturally occurring trait variations can not easily be put into two groups. Trees in a forest are not neatly divided into categories of "short" and "tall" any more than a population of birds separates cleanly into "light" and "heavy." Instead, traits like size, color, and shape often have subtle and continuous variation throughout the population. Nearly every aspect of any species displays individual differences of this nature (Falconer and Mackay, 1989). These subtle variations were the ones Darwin himself referred to as the raw material upon which natural selection can act (Darwin, 1859). *Quantitative genetics* is the branch of genetics dedicated to understanding how genetic and environmental factors shape these quantitative traits (Falconer and Mackay, 1989; Griffiths et al., 2004). A key goal within the field is to partition the phenotypic variance – the variation in a trait within a population – into its genetic and environmental components. This decomposition enables us to estimate *heritability*, the proportion of total phenotypic variance attributable to genetic factors, and *additive genetic effects*, which in sum describe the genetic contribution to the deviation from the population mean for a trait of an individual. These estimates have enabled breeders and biologists alike to predict response to selection. Classical methods typically relied on pedigrees to infer genetic relationships, which for some species can be challenging and time-consuming to construct. Additionally, pedigree-based relatedness estimates are based on expected values and do not reflect true genetic relatedness.

Advances in molecular biology – particularly genome sequencing and genotyping – have paved the way for *genomic prediction*, an extension of the quantitative genetics field that leverages genome-wide marker data to predict the genetic contribution to phenotypic traits (McGaugh et al., 2021; Barreto et al., 2024). Rather than inferring relationships indirectly through pedigrees, genomic prediction uses genetic markers such as *single nucleotide polymorphisms* (SNPs) to precisely estimate how closely related individuals are at the DNA level. Besides enhancing relatedness estimates, access to genetic markers enables direct modeling of genetic effects, opening the door to a range of predictive methods. The shift to genomic prediction allowed for more accurate predictions and has revolutionized the field of breeding (Meuwissen et al., 2016).

While extensively used in animal and plant breeding, genomic prediction was not until more recently recognized within the fields of ecological and population genetics (McGaugh et al., 2021; Ashraf et al., 2021; Hunter et al., 2022). Here, genomic prediction provides insights into the genetic architecture of traits, helps track micro-evolutionary processes, and supports planning of conservation strategies (McGaugh et al., 2021; Ashraf et al., 2021). However, prediction in wild populations has proved to be much more challenging than in controlled circumstances. In contrast to livestock selectively bred under relatively uniform environmental conditions promoting homogeneity, wild populations experience natural selection among diverse environmental factors that shape their traits. This variability makes model generalization challenging, especially when predicting across different subpopulations. As a result, there is a growing need for new approaches and innovations in the field of genomic prediction.

Classical linear models, such as the *animal model*, have been the cornerstone in partitioning variance and estimating additive effects. However, they may overlook non-linear

effects that contribute to trait variation. In contrast, there are modern methods originating from the field of *machine learning* designed to address such effects. Two prominent classes of machine learning models are *neural networks* and *decision trees*. Unlike traditional linear models, these are not constrained to linear relationships. Instead, they are designed to capture complex interactions and non-linearities, and both have become central tools in predictive modeling across various domains. Another influential advancement within machine learning is *gradient boosting*. Since its formal introduction in 2001, gradient boosting has been an undeniable force (Friedman, 2001). Combined with decision trees, it forms the basis for several state-of-the-art algorithms (Chen and Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2017) that have the ability to capture aspects of genetic variation that are not accounted for by the linear models currently used in genomic prediction. Despite their potential, existing studies benchmarking gradient boosted decision trees against linear approaches in genomic prediction have produced mixed results, not consistently demonstrating superior performance (Li et al., 2018; Gill et al., 2022; Lourenço et al., 2024; Barreto et al., 2024). Nevertheless, gradient boosted decision trees remain relatively unexplored in the context of wild populations, where the added complexity of natural environments may favor flexible, non-linear approaches over linear methods. However, gradient boosted decision trees present their own set of challenges. They often rely on extensive *hyperparameter tuning* to optimize performance and prevent *overfitting* – a common pitfall when working with thousands of variables. To mitigate such issues, methods for dimension reduction, such as *subset selection* or *feature extraction*, can be employed. These techniques reduce computational complexity and potentially enhance predictive accuracy on unseen data (Bengio et al., 2013; James et al., 2023).

This thesis seeks to determine if advanced machine learning approaches can compete with linear models in genomic prediction. Picking up the thread from Wold (2024), we built upon previous experiments and explored new approaches. Specifically, we investigated how the number of variables affects model accuracy and whether compressed representations of genetic data can effectively replace raw markers. Initially, we performed a visual exploratory analysis to detect possible clustering patterns reflecting genetic structure among individuals. For this analysis, we employed *t-distributed stochastic neighbor embedding* (t-SNE), a tool for visualizing high-dimensional data. Following this visualization, we conducted a subset selection experiment, evaluating model performance as the number of genetic markers was incrementally increased. The aim was to identify an optimal subset size – if present – that maintains or improves accuracy compared to using the full marker set, thereby also reducing computational effort in future work. We trained models on randomly sampled subsets of varying sizes to assess the relationship between marker quantity and predictive accuracy. Using the optimal subset, we subsequently applied three gradient boosting frameworks – XGBoost, LightGBM, and CatBoost – for genomic prediction, performing hyperparameter tuning to maximize accuracy. CatBoost and LightGBM were fitted with decision trees as base learners, while XGBoost was fitted with both a linear and a tree-based base learner. These models were benchmarked against two established approaches: a Bayesian animal model, implemented using integrated nested Laplace approximation (INLA), representing the gold standard, and *ridge regression*, a lightweight, scalable linear method. Lastly, we explored the use of linear and non-linear feature extraction methods, namely *principal component analysis* (PCA) and a neural-network-based *autoencoder*, to create compressed feature sets to use as inputs for gradient boosted trees. All performance metrics reported in this thesis are obtained through cross-validation, ensuring they are representative of general model performance. All code necessary to reproduce the results is openly available in the GitHub repository simenklw/MasterThesis.

The dataset used in this thesis was provided by the Department of Biology at NTNU and the Gjærevoll Centre. It includes SNP genotypes, phenotypic measurements, and environmental data from a long-term study of wild house sparrows inhabiting an island system off the coast of northern Norway. This population has been monitored continuously since 1993, providing a comprehensive dataset well suited to study genomic prediction in a natural setting (Ranke et al., 2018). For the purposes of this study, all individuals are pooled and treated as a single population. However, known migration patterns and environmental heterogeneity between islands influence the population's genetic structure, adding complexity to the prediction task. We focus on three quantitative traits: body mass, tarsus length, and wing length.

The thesis is organized as follows: The background section introduces key biological and machine learning concepts. The methods section details our procedures and data processing steps. The results section presents relevant findings, and finally, we discuss the implications of these findings, their relevance to genomic prediction in wild populations, and potential directions for future research.

Improving genomic prediction in wild populations enhances our understanding of the relationship between genes and phenotypes. Insights into genetic architecture can help anticipate and interpret how populations respond to environmental change. Such knowledge is valuable for conservation efforts and maintaining biodiversity. Thereby, this thesis contributes to goals 13 (climate action) and 15 (life on land) of the United Nations Sustainable Development Goals (United Nations, 2015).

Note to reader: This thesis is a direct continuation of the work presented in Wold (2024). It builds upon the same theoretical foundation and reuses the same data processing pipeline. As a result, several passages in Chapters 1, 2, and 3 contain identical or nearly identical wording from Wold (2024). Additionally, the results from the animal model and the t-SNE analysis are carried over from Wold (2024).

2 Background

2.1 Fundamentals of Quantitative Genetics

2.1.1 Basic Genetics

Before going into the methodology and finer details, it is essential to establish a foundational understanding of genetics. In simple terms, genetics is the study of genes. Genes are segments of deoxyribonucleic acid (DNA), the hereditary material passed from parents to offspring (Griffiths et al., 2004). Genes play a crucial role in determining the fundamental characteristics of a species, and variation within a species arises from differences in genetic makeup. A gene can exist in different forms, known as alleles, which may vary slightly or significantly from one another. Each gene allele occupies a specific position along a chromosome, known as the gene locus (plural: loci).

When a particular characteristic within a population occurs in two or more distinct forms, it is called a phenotype (Griffiths et al., 2004). Examples of phenotypes include traits like eye color or blood type, with different phenotypes being "blue eyes" versus "brown eyes" or "type O" versus "type A" blood. The phenotypic traits are determined by the alleles an individual carries. In biparental organisms, two alleles – one from the mother and one from the father – are inherited. These alleles may be identical or different, but they make up the individual's genotype together, which can partially or fully determine their phenotype. Although rare, certain traits are determined by only one gene and exhibit distinct phenotypes. For example, a plant might have either purple or white flowers with no intermediate colors. These are called Mendelian traits. However, most traits are polygenetic, meaning multiple genes influence them, and typically display continuous variation throughout a population, resulting in a range of phenotypes rather than discrete categories, as discussed in the introduction. They can also be heavily influenced by the environment and even interactions between genes and the environment (Falconer and Mackay, 1989).

2.1.2 Phenotypic Variation and Heritability

Phenotypic variation refers to the individual differences in observable traits within a population. These differences arise from genetic factors (DNA) and environmental influences (*e.g.*, climate, diet, habitat). The phenotype (P) can be represented as the sum of the genetic effect (G) and the environmental effect (E), expressed as $P = G + E$. Assuming that G and E are independent, the total phenotypic variance V_P can be partitioned into $V_P = V_G + V_E$, where V_G is the genetic variance and V_E is the environmental variance (Falconer and Mackay, 1989). The genetic component G of the phenotype P can be further broken down into subcomponents $G = A + D + I$, where A represents the additive genetic effect, D is the dominance deviation, and I is the interaction deviation. Since each subcomponent contributes to genetic variation, the overall genetic variance V_G can be expressed as $V_G = V_A + V_D + V_I$. The dominance deviation D reflects the interaction between alleles at a single locus, where the combined effect of alleles is not simply the sum of their individual effects. The interaction deviation I, on the other hand, captures the aggregate interactions across multiple loci. The additive genetic effect A is the sum of the independent contributions of each allele to the phenotype (Falconer and Mackay, 1989; Lynch and Walsh, 1998; Wray et al., 2019). Apart from telling us how much of the

genetic variation is due to additive effects, V_A is also directly related to the *narrow-sense heritability*, h^2, formally defined $h^2 = \frac{V_A}{V_P}$ (Lynch and Walsh, 1998; Conner and Hartl, 2004; Visscher et al., 2008). The narrow-sense heritability quantifies with a single number the fraction of variation between individuals in a population that is due to the additive effect of their genotypes. High h^2 indicates that additive genetic factors play a substantial role in trait differences, enabling more reliable predictions about responses to selection (Falconer and Mackay, 1989). Narrow-sense heritability estimates are valuable in fields from livestock breeding to human medicine, where they help disentangle genetic and environmental contributions to traits and diseases (Visscher et al., 2008). Having introduced these foundational concepts, we now turn to discuss some of the classical approaches used to model genetic effects and their associated variance.

2.2 From Pedigree-Based to Genomic Prediction

2.2.1 The Animal Model and the Shift to Genomic Methods

Linear mixed models (LMMs) used to predict genetic parameters in plants and animals are usually referred to as animal models (Lynch and Walsh, 1998; Kruuk, 2004). Originally, these models served as a means to partition phenotypic variance (Mrode, 2014), but we will see that they can also provide *breeding values*. In its simplest form, an animal model can be expressed as

$$y_i = \mu + a_i + \varepsilon_i \qquad i = 1, ..., N \ , \tag{1}$$

where y_i is the phenotype of individual i, μ the overall population mean, a_i the additive genetic effect (breeding value) of individual i, and ε_i is the individual environmental effect for individual i, assumed to be independently and normally distributed as $\varepsilon_i \sim N(0, V_E)$. The vector of breeding values $\boldsymbol{a} = (a_1, \ldots, a_N)^T$ is assumed to follow a multivariate normal distribution $\boldsymbol{a} \sim N(\mathbf{0}, V_A\mathbf{G})$, where $\mathbf{G}$ is a *relatedness matrix*, encoded with the relatedness between individuals, and V_A is the additive genetic variance. The breeding values are extracted from the model via the *best linear unbiased predictor* (BLUP, Henderson, 1984), as shown in Section 3.2.

Breeding values represent an individual's additive genetic merit. Historically, before genomic data were available, breeding values were inferred from pedigree information and phenotypes. The breeding value of an individual for a trait can be thought of as the total additive genetic value it contributes, half of which is transmitted to its offspring on average (Lush, 1943; Falconer and Mackay, 1989; Kliman, 2016). For instance, given the breeding values for a given trait of a cow and a bull, one can compute the expected phenotype of their offspring. In principle, if we knew the effect of every allele at every locus, we could compute the exact breeding value $BV = \sum_i b_i x_i$, where b_i is the average effect of the i^{th} allele, and $x_i \in \{0, 1, 2\}$ is the genotype for the trait-increasing allele (Wray et al., 2019). In reality, we do not know the exact causal loci or their magnitude, so breeding values must be estimated.

The coefficients of the pedigree-based relatedness matrix $\mathbf{G}$ reflected expected genetic relatedness. For example, if individual i and j were full siblings, $\mathbf{G}_{ij} = 0.5$, indicating that, on average, half of their alleles were identical by descent. With the advent of genomic data, the old pedigree-based $\mathbf{G}$ could be replaced by a *genomic relationship matrix* (GRM), constructed from genome-wide marker data. By using genetic markers instead of

pedigrees, we measure the actual genetic similarity between individuals rather than relying on expected values from their ancestral lineages. When two siblings are genotyped, their genomic similarity might deviate from the expected 0.5, reflecting the actual proportion of shared alleles. This shift led to *genomic best linear unbiased predictions* (GBLUPs), where the animal model leverages genomic information for more accurate estimates of breeding values.

Relatedness Estimated from Single Nucleotide Polymorphisms (SNPs)

To build the GRM and implement genomic approaches, we can use SNPs as genetic markers (Visscher et al., 2008). A SNP (Figure 1) is a variation at a single nucleotide position in the DNA sequence (Jehan and Lakhanpaul, 2006; National Library of Medicine, 2022; International Society of Genetic Genealogy, 2024). Such variants are abundant throughout the genome and are typically biallelic, meaning they have only two common variants (alleles) in the population (Jehan and Lakhanpaul, 2006; International Society of Genetic Genealogy, 2024). This biallelic nature simplifies data encoding. For example, consider a SNP with alleles A (reference) and G (alternate). The three possible genotypes (AA, AG, GG) can be numerically encoded as (0, 1, 2) to represent the count of alternate alleles. This straightforward encoding allows statistical models to incorporate genomic information efficiently.

Figure 1: The SNP genotype refers to the specific combination of alleles an individual carries at a particular gene or SNP location. Since we inherit two copies of each basepair (one from each parent), the SNP genotype can be homozygous (AA or GG) or heterozygous (AG), as shown here.

Due to *linkage disequilibrium* (LD), the non-random association of alleles at different loci (Lynch and Walsh, 1998; Conner and Hartl, 2004), SNPs serve as proxies for causal variants (the genetic variation that actually causes phenotypic variation). SNPs physically close to a causal variant tend to be inherited together with that variant, making them correlated. As a result, SNPs can indirectly capture the effect of causal variants even if we do not know their exact location or effect size. By including a large number of SNPs spanning the entire genome, we increase the likelihood that some of these markers are in LD with causal variants. As a result, we can model genetic contribution to phenotypes directly from SNP data. Regarding the GRM, it can be constructed from SNPs via the following method proposed by VanRaden (2008)

$$\mathbf{G} = \frac{\mathbf{Z}\mathbf{Z}^{\mathbf{T}}}{2\sum_{i=1}^{p} p_i(1-p_i)} \ ,$$

where $\mathbf{Z}$ is derived from a SNP-matrix and p_i represents an allele frequency at locus i. In this thesis, we will use an extension of the simple animal model displayed in equation (1), including environmental variables, with a GRM to obtain GBLUP estimates of breeding values for different traits. These estimates will serve as a benchmark when evaluating the performance of other predictive models in this thesis.

2.2.2 Marker-Based Regression

Another popular approach to modelling breeding values is *marker-based regression*. While GBLUP aggregates genomic information through the GRM, marker-based regression models the breeding values as an explicit function of all genetic markers (De Los Campos et al., 2010). The approach stems from the observation that quantitative traits, such as body mass or wing length, result from the combined contributions of many loci genome-wide and environmental factors (Griffiths et al., 2004; Walsh and Lynch, 2018). The marker-based linear model uses p SNPs for N individuals and assumes that the phenotype of each individual can be decomposed as a linear combination of contributions from markers, along with other fixed and random effects. A marker-based regression model can be formulated as

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{X}\mathbf{b} + \mathbf{Z}\mathbf{u} + \mathbf{W}\mathbf{d} + \boldsymbol{\varepsilon} \ , \tag{2}$$

where $\mathbf{y}$ is the vector of phenotypic observations, $\boldsymbol{\mu}$ is the overall population mean, $\mathbf{X}\mathbf{b}$ represents the fixed effects (*e.g.*, sex, age), $\mathbf{W}\mathbf{d}$ accounts for random environmental effects, $\mathbf{Z}$ is the SNP matrix containing allele counts, $\mathbf{u}$ are the SNP effect sizes, and $\boldsymbol{\varepsilon}$ is the residual error term. In equation (2) $\mathbf{Z}\mathbf{u}$ captures the total additive genetic effects as the sum of contributions from all markers, effectively forming the breeding values.

Although conceptually simple, this linear formulation encounters two potential problems in genomic data. First, high dimensionality ($N \ll p$) makes the ordinary least squares solution ill-posed and prone to extreme overfitting. Second, both the animal model and the marker-based linear regression as posed in equation (2) assume purely additive, linear effects and may miss non-linear ones. These limitations motivate the use of techniques that either stabilise the linear model or move beyond linearity altogether.

A classical linear solution to the overfitting problem is *ridge regression* (RR), already proposed for genomic prediction (GP) by Whittaker et al. (2000). RR stabilises prediction, is computationally light and highly scalable, and under specific parameters becomes mathematically equivalent to GBLUP (Habier et al., 2007). It is expected to perform particularly well when genetic effects are spread thinly over many loci rather than concentrated in a few large-effect SNPs (De Vlaming and Groenen, 2015). We therefore adopt RR as a fast and simple alternative against which to weigh the added complexity of the Bayesian approach that we take to fit the animal model. The full formulation of RR is presented in Section 2.3.7. As a linear model, however, it remains limited to modeling linear relationships. To capture both large feature spaces and potential non-linear patterns, we investigate more flexible techniques. Both for direct marker-based regression and for visualising our data.

2.3 Machine Learning Methods

2.3.1 Introduction and Motivation

While the assumption of independence between genetic and environmental effects (see Section 2.1.2) is common in statistical applications for GP, a more nuanced model would include so-called gene-by-environment ($G \times E$) interactions (Falconer and Mackay, 1989). These interactions imply that individuals with identical genetic makeup may develop markedly different traits if exposed to distinct environments. In livestock, controlled housing, standardized feeding regimes, and disease prevention keep the environmental share of phenotypic variance small. Because of that, models based purely on genome-wide markers might explain the phenotypic variance well. Wild populations are different. Individuals may experience heterogeneous climates, diets, pathogens, and social conditions, meaning a larger fraction of variance comes from the environment and potential $G \times E$ interactions. As a result, predicting phenotype from genetic data alone is fundamentally harder in the wild. Although modelling $G \times E$ is desirable, it would require a single model that ingests both genetic markers and measured environmental variables. A decision tree or a neural network might handle this challenge, but for a linear model, that would mean adding thousands of interaction terms. In this thesis, we instead adopt a pragmatic two-step strategy, adjusting the true phenotype for non-genetic effects, as in Ashraf et al. (2021); Hunter et al. (2022). First, we regress the phenotype on measured environmental variables and remove their effects, then we model the residual effects from the genetic markers. But even after this adjustment, significant challenges still remain. There might be non-linear genetic effects that purely linear models, such as RR, cannot capture without an explosion of interaction terms. In addition, our sparrow data may show population structure – island sub-populations with distinct allele frequencies and LD patterns. Population structure is an issue in genetic studies, leading to population-specific patterns of association between genotypes and phenotypes (Lasky-Su et al., 2008; Hellwege et al., 2017; Zaidi and Mathieson, 2020). Modern machine learning (ML) algorithms offer potential solutions. With their flexible architectures, tree-based models and neural networks are specifically designed to capture non-linear, complex patterns that linear models may miss.

Before detailing specific algorithms, it is useful to view ML more broadly. Machine learning, a term coined in 1959 (Samuel, 1959), emerged from artificial intelligence research and overlaps substantially with statistical learning (SL), both inherently focused on learning from data. Whereas SL traditionally has emphasised interpretability and uncertainty quantification, ML prioritises predictive accuracy and computational scalability. In essence, ML seeks a function $\boldsymbol{f}(\mathbf{x}) = y$ that generalises well beyond the training sample. Given the paired data $\{y_i, \mathbf{x}_i\}_{i=1}^{N}$, it minimizes the expected prediction error, $(y - f(\mathbf{x}))$, of a new and unseen pair $(y, \mathbf{x})$ (Domingos, 2012; Fernandes de Mello and Antonelli Ponti, 2018). In this thesis, we use ML in two ways – for predicting the genetic component of phenotypes and for representation learning, compressing the vast SNP matrix into informative low-dimensional features. The rest of Chapter 2 presents the theory and ideas behind the methods we employ.

2.3.2 Regression Trees

Regression trees are a variation of decision trees used to predict continuous outcomes. They learn patterns by recursively partitioning the data based on input features. The underlying idea is to find the best "questions" to ask so that the data is split into subsets

where the target values are as similar as possible. Imagine we have a set of observations, each with a target value and feature values. The tree starts with the full dataset and seeks the best possible question to ask. For continuous features, such a question takes the form, "Is feature $x_j < s$?" where s is some threshold value. Having found the best possible question, the tree makes a decision rule based on one of the input features where the tree splits the data into two subsets, one where $x_j < s$ and one where $x_j \geq s$. A simple regression tree is depicted in Figure 2.

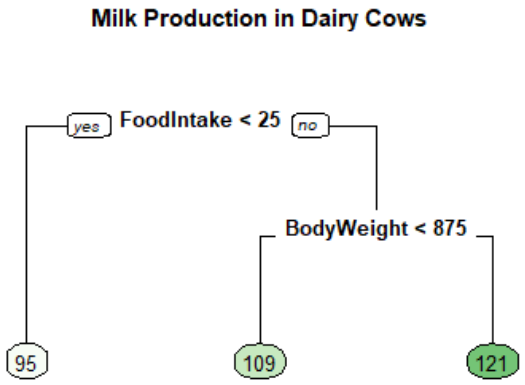


Figure 2: Graphical representation of a regression tree. In this model, milk production in dairy cows is regressed on FoodIntake and BodyWeight. The tree represents the fitted decision structure, starting by splitting in FoodIntake, then BodyWeight if FoodIntake is more than 25. The numbers in the three leaf nodes are the predictions for a new observation ending up in the respective leaf node. The plot was created for illustrative purposes only; hence, the numerical values are not representative.

As described in Hastie et al. (2009) and James et al. (2023), the goal is to find regions $R_1, \ldots, R_J$ that minimize the loss $\mathcal{L}$ over all regions. Each region is assigned a prediction $\hat{y}_{R_j}$, typically based on a statistic of the target values in region R_j, such as the average. This prediction becomes the estimated target value for any observation in that region. The total loss is then calculated as

$$\sum_{j=1}^{J} \sum_{i \in R_j} \mathcal{L}(y_i, \hat{y}_{R_j}) \ ,$$

where y_i is the true target value for observation i, and $\hat{y}_{R_j}$ is the predicted value for region R_j. However, considering all possible combinations of splits is computationally infeasible. Therefore, we employ a greedy, top-down approach, choosing recursively the best split at that particular step and not looking ahead. We start by dividing the feature space into $R_1(j,s) = \{x | x_j < s\}$ and $R_2(j,s) = \{x | x_j \geq s\}$ by splitting one of the features $x_1, x_2, \ldots, x_p$. Thus we need to find the feature j and splitting value s that minimize

$$\sum_{i: x_i \in R_1(j,s)} \mathcal{L}(y_i, \hat{y}_{R_1}) + \sum_{i: x_i \in R_2(j,s)} \mathcal{L}(y_i, \hat{y}_{R_2}) \ ,$$

where $\hat{y}_{R_i}$ is the mean response for the training observations in $R_i(j,s)$. This process is

repeated recursively for each new region to further reduce the loss within them until a stopping criterion is met, such as a minimum number of observations in each region or a maximum tree depth. The process is outlined in Algorithm 1.

Algorithm 1 Regression Tree

Initialize the root node with all training data.
while stopping criteria not met **do**
    for all features x_j and possible split points s **do**
        Split the data into $R_1(j, s)$ and $R_2(j, s)$
        Compute the loss reduction
    end for
    Select the feature x_j and split point s that result in greatest loss reduction.
    Split the data into two child nodes based on the selected split.
    Repeat steps 3-8 for each new node.
end while
Output the final tree with predictions at each terminal node.

While linear models may have an advantage if the relationship between the target and feature variables is truly linear, regression trees are capable of capturing non-linear and complex relationships (Hastie et al., 2009). A single tree also has the advantage of being easy to interpret (Figure 2). However, in practice, using a single tree may not provide the best predictive performance. Therefore, we often use an ensemble of trees to improve accuracy, which will be discussed in the next section.

2.3.3 Boosting and Gradient Boosting

To build up to more advanced algorithms, we first need to understand *boosting* and *gradient boosting*. The following description and algorithm are based on Hastie et al. (2009). Gradient boosting is a general-purpose framework capable of solving regression, classification, and ranking problems, with industry-level applications that can often be used straight out of the box. Due to its versatility and efficiency, gradient boosting has become the foundation for a variety of algorithms achieving state-of-the-art performance, some of which we will make use of in this project.

A key term we will use frequently going forward is *weak learner* (also known as *base learner*). In ML, a weak learner is a model that performs slightly better than random guessing for classification tasks or predicting the mean for regression tasks (Bishop, 2006; James et al., 2023). Despite their limited individual performance, weak learners can improve significantly when bundled together. In regression with a weak learner, such as linear regression or a shallow regression tree, we aim to learn a prediction function from our features

$$\underbrace{\boldsymbol{X}}_{Features} \longrightarrow \underbrace{f(\boldsymbol{X})}_{Prediction function} .$$

Methods employing a combination of weak learners to create more powerful models are called *ensemble methods*. With boosting, we learn the prediction function F as a sum of M weak learners

$$F(\boldsymbol{X}) = \sum_{t=1}^{M} f^{(t)}(\boldsymbol{X}) \ ,$$

where $f^{(t)}(\boldsymbol{X})$ is the t^{th} weak learner. Given a dataset of paired values $\{y_i, \mathbf{x}_i\}_{i=1}^N$, where y_i represents the target values and $\mathbf{x}_i$ represents the input features, the goal of boosting is again to minimize a loss function $\mathcal{L}(y_i, \hat{y}_i)$, where $\hat{y}_i$ is the prediction for y_i. Unlike parallel training methods, boosting trains sequentially, with each weak learner aiming to correct the errors of the previous one. A straightforward way to achieve this is by fitting a new base learner directly to the errors of the previous one and adding that learner to the ensemble. Gradient boosting, however, takes a different approach by incorporating ideas from the optimization procedure known as *gradient descent*, where we move in the direction that fastest minimizes the loss function. Instead of fitting to the residuals directly, gradient boosting fits a new weak learner to the negative gradients of the loss function. We start by defining a loss function $\mathcal{L}$, which needs to be differentiable so we can compute its gradients. At step t, with the current prediction function $F^{(t)}$, we calculate the negative gradients

$$r_i^{(t)} = - \left. \frac{\partial \mathcal{L}(y_i, \hat{y}_i^{(t)})}{\partial \hat{y}_i^{(t)}} \right|_{\hat{y}_i^{(t)} = F^{(t)}(\mathbf{x}_i)} \qquad i = 1, \ldots, N \ .$$

Next, we generate a new dataset $\mathcal{D} = \{(r_i^{(t)}, \mathbf{x}_i) : i = 1, 2 \ldots N\}$, where the input features remain the same but the target values are now the negative gradients $\boldsymbol{r}^{(t)}$. We train a new base learner $f^{(t+1)}$ to predict these negative gradients. A portion of this learner is then added to the existing model $F^{(t)}$. Writing it out

$$F^{(t+1)}(\mathbf{x}) = F^{(t)}(\mathbf{x}) + \gamma^{(t)} f^{(t+1)}(\mathbf{x}) \approx F^{(t)}(\mathbf{x}) - \gamma^{(t)} \nabla \mathcal{L}(y, F^{(t)}(\mathbf{x})) \ ,$$

we see that this is, in fact, an approximation of gradient descent, where γ is the step size, determining how much of the new base learner is added to the ensemble. In the field of ML, this γ is usually a global parameter called *learning rate*, applied to all base learners, and is set as a parameter by the user. However, one could also optimize $\gamma^{(t)}$ at each step by performing a line search where

$$\gamma^{(t)} = \underset{\gamma}{arg\ min} \left[\sum_{i=1}^{N} \mathcal{L}(y_i, F^{(t)}(\mathbf{x}_i) + \gamma f^{(t+1)}(\mathbf{x}_i)) \right] \qquad i = 1, \ldots, N \ .$$

These steps are repeated iteratively until we have built M weak learners. The complete procedure is summarized in Algorithm 2.

In the remainder of Section 2.3, we will explore the three gradient boosting frameworks utilized in this project, XGBoost, LightGBM, and CatBoost. We will begin with XGBoost, as it established the foundation for the others, and proceed in the order of their release. Since these frameworks share common underlying mathematical principles, we will provide detailed explanations using XGBoost and then highlight the key differences and unique features of LightGBM and CatBoost in plain language. Following that, we briefly introduce RR and linear regression as a base learner. While tree-based models are our primary tool in this project, we will also run linear models. Lastly, we will provide an overview of *regularization*, *hyperparameters*, and *hyperparameter optimization*, which are crucial concepts in ML and essential for building effective models.

Algorithm 2 Gradient Boosting

Initialize model with a constant value: $F^{(1)}(\mathbf{x}) = \arg\min_c \sum_{i=1}^{N} \mathcal{L}(y_i, c)$
for $t = 1, 2, \ldots, M$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    Compute negative gradients: $r_i^{(t)} = -\left.\frac{\partial \mathcal{L}(y_i, \hat{y}_i^{(t)})}{\partial \hat{y}_i^{(t)}}\right|_{\hat{y}_i^{(t)} = F^{(t)}(\mathbf{x}_i)}$
  end for
  Fit a base learner $f^{(t+1)}$ to the residuals $\boldsymbol{r}^{(t)}$: $f^{(t+1)} = \text{TrainBL}\left(\{r_i^{(t)}, \mathbf{x}_i\}_{i=1}^{N}\right)$
  if optimizing $\gamma^{(t)}$ **then**
    Compute $\gamma^{(t)} = \arg\min_\gamma \left[\sum_{i=1}^{N} \mathcal{L}(y_i, F^{(t)}(\mathbf{x}_i) + \gamma f^{(t+1)}(\mathbf{x}_i))\right]$
  end if
  Update model: $F^{(t+1)}(\mathbf{x}) = F^{(t)}(\mathbf{x}) + \gamma^{(t)} f^{(t+1)}(\mathbf{x})$
end for
Output: $F^{(M)}(\mathbf{x})$

2.3.4 XGBoost

Since its release in 2014, XGBoost (Extreme Gradient Boosting, Chen and Guestrin, 2016) has had a profound impact on the ML community. Ten years later, it is still the go-to algorithm for a wide range of problems, commonly used for both regression and classification tasks. As the name suggests, it builds on gradient boosting, where weak learners are combined to form a more powerful model.

XGBoost uses a more complex objective function, including both the loss function from Section 2.3.3 and a regularization term, a key detail in XGBoost that penalizes complex models and encourages simplicity. The general function can be expressed as

$$L^{(t)} = \sum_{i=1}^{n} \mathcal{L}(y_i, \hat{y}_i^{(t)}) + \Omega(f^{(t)}) \ , \tag{3}$$

where $\mathcal{L}(y_i, \hat{y}_i^{(t)})$ is the loss function and $\Omega(f^{(t)})$ is a regularization term. The regularization term Ω can have the following form

$$\Omega(f^{(t)}) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \ , \tag{4}$$

but can also be expanded to include additional terms. In equation (4), T is the number of leaves in the tree, w_j are the leaf weights, and γ and λ are regularization parameters. When Taylor expanding the loss function in equation (3) to the second order, we obtain

$$\mathcal{L}(y_i, \hat{y}_i^{(t-1)} + f^{(t)}(\mathbf{x_i})) \approx \mathcal{L}(y_i, \hat{y}_i^{(t-1)}) + g_i f^{(t)}(\mathbf{x}_i) + \frac{1}{2} h_i f^{(t)}(\mathbf{x}_i)^2 \ , \tag{5}$$

where $g_i = \frac{\partial \mathcal{L}(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ and $h_i = \frac{\partial^2 \mathcal{L}(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$, the first- and second-order derivatives of the loss function. Since $\mathcal{L}(y_i, \hat{y}_i^{(t-1)})$ is constant regardless of the choice of $f^{(t)}$, we can remove it and work with an easier expression going forward, giving us

$$\tilde{L}^{(t)} = \sum_{i=1}^{n} \left[g_i f^{(t)}(\mathbf{x}_i) + \frac{1}{2} h_i f^{(t)}(\mathbf{x}_i)^2 \right] + \Omega f^{(t)} \ .$$

Now, we do a trick that allows us to directly compute the optimal weights for any tree structure with T leaf nodes. Let I_j be the set of instances i that are in the j^{th} leaf node of $f^{(t)}$, allowing the reformulation

$$\tilde{L}^{(t)} = \sum_{j=1}^{T} \left[\sum_{i \in I_j} g_i f^{(t)}(\mathbf{x}_i) + \frac{1}{2} \sum_{i \in I_j} h_i f^{(t)}(\mathbf{x}_i)^2 \right] + \Omega(f^{(t)}) \ .$$

For all instances i in I_j, the new tree yields the same prediction value $f^{(t)}(\mathbf{x}_i) = w_j$. Swapping $f^{(t)}(\mathbf{x}_i)$ with w_j and writing out $\Omega(f^{(t)})$ gives us

$$\tilde{L}^{(t)} = \sum_{j=1}^{T} \left[\sum_{i \in I_j} g_i w_j + \frac{1}{2} \sum_{i \in I_j} h_i w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \ ,$$

which we can rearrange into

$$\tilde{L}^{(t)} = \sum_{j=1}^{T} \left[w_j \sum_{i \in I_j} g_i + \frac{1}{2} w_j^2 \left(\sum_{i \in I_j} h_i + \lambda \right) \right] + \gamma T \ . \tag{6}$$

We see that for each leaf node j, the objective function is quadratic in w_j. To find the optimal weights, we take the derivative $\frac{\partial \tilde{L}^{(t)}}{\partial w_j}$, equate it to zero and solve for w_j, which yields

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \ . \tag{7}$$

Inserting (7) into (6) we obtain

$$\tilde{L}^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \ .$$

Having an expression for the optimal weights and loss for a tree with T leaf nodes, we can consider different splits by comparing the objective before and after a split. We look for the split that gives the largest reduction in the objective function. Let I be the set of n instances in the current node, and I_L and I_R are the set of instances that fall into the left and right child nodes, respectively. We compute the loss reduction as follows:

$$L_{\text{before split}} = -\frac{1}{2} \frac{\left(\sum_{i \in I} g_i \right)^2}{\sum_{i \in I} h_i + \lambda} + \gamma \ ,$$

$$L_{\text{after split}} = L_L + L_R = -\frac{1}{2} \frac{\left(\sum_{i \in I_L} g_i \right)^2}{\sum_{i \in I_L} h_i + \lambda} - \frac{1}{2} \frac{\left(\sum_{i \in I_R} g_i \right)^2}{\sum_{i \in I_R} h_i + \lambda} + 2\gamma \ .$$

The reduction in the objective function is then

$$\Delta L = L_{\text{before split}} - L_{\text{after split}} = L - (L_L + L_R) \ ,$$

$$\Delta L = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i \right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i \right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i \right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \ .$$

XGBoost will choose the split with the highest gain in a level-wise manner, stopping the iteration if the gain is negative or too small.

To sum up, XGBoost enhances traditional gradient boosting by using a second-order Taylor expansion of the loss function, approximating it with both gradients and Hessians. This approach allows for more efficient optimization by directly calculating optimal leaf weights and selecting splits that maximize objective function reduction, with trees constructed in a level-wise manner until additional splits yield minimal gain.

2.3.5 LightGBM

LightGBM (Light Gradient Boosting Machine, Ke et al., 2017) is also a powerful gradient boosting framework. The main ideas from the previously explained XGBoost are retained, but there are important differences that we will outline here in plain text. The algorithm is developed by Microsoft and optimized for both speed and accuracy. It achieves this by implementing two innovative techniques, Gradient-Based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), which have the potential to significantly reduce the time and computational power needed to find optimal splits for the base learners. Additionally, LightGBM uses histogram-based binning (also supported by XGBoost), where continuous values are discretized into bins, reducing memory usage and further speeding up training.

Another key feature is LightGBM's leaf-wise growth strategy. Unlike the level-wise growth used by XGBoost (Chen and Guestrin, 2016), which expands all leaves to the same depth before moving deeper, LightGBM grows trees leaf-wise, meaning it always splits the leaf node with the highest potential gain. This approach allows LightGBM to create asymmetrical trees that focus on regions of the data where greater precision is needed, often resulting in a faster reduction in loss.

The real game changers in LightGBM are GOSS and EFB. GOSS helps LightGBM prioritize data with larger gradients (indicating higher error). When subsampling data to build new base learners, GOSS makes sure the data that we can learn the most from is retained, while dropping data points with smaller gradients. This effectively reduces the number of data points required per base learner, without significant sacrifices in accuracy. EFB, on the other hand, bundles mutually exclusive features (those that rarely have non-zero values simultaneously) into a single feature, which reduces dimensionality. When a large number of mutually exclusive features are present, EFB can contribute to a substantial speedup. More on GOSS and EFB can be read in Ke et al. (2017).

Together, these design elements – GOSS, EFB, histogram-based binning, and leaf-wise growth – make LightGBM particularly well-suited for large-scale tasks where both accuracy and computational efficiency are essential. Ke et al. (2017) conclude that LightGBM is capable of significantly outperforming XGBoost regarding computational speed and memory usage while maintaining high accuracy.

2.3.6 CatBoost

The final gradient boosting algorithm we will explore is CatBoost (Categorical Boosting, Prokhorenkova et al., 2017), developed by the Russian technology company Yandex in 2017 as yet another gradient boosting algorithm that uses decision trees as its base learner.

While CatBoost shares foundational aspects with XGBoost and LightGBM, utilizing both gradients and Hessians to evaluate gain for possible splits, CatBoost introduces two critical advances that we will further explore, *ordered boosting* and *ordered target encoding*.

Both ordered boosting and ordered target encoding were created to combat the *prediction shift* caused by the *target leakage* present in alternative algorithms (Prokhorenkova et al., 2017). Target leakage is referred to as the use of information in training that would not be available when making predictions on new data, leading to overly optimistic results. An example of leakage would be to include a feature like *minutesLate* when predicting whether someone arrives late. However, the value of *minutesLate* would only be available after the person has arrived in the first place. While this might seem obvious, the leakages that CatBoost addresses are more subtle. The first one arises from *target encoding*, an effective method for preprocessing categorical features in ML, where a categorical feature value is replaced by some number derived from the target, leading to bias (Prokhorenkova et al., 2017). As the target value for a data point influences the encoding of the same point that we want to predict the target with. The second type of leakage occurs from the boosting procedure itself. In standard gradient boosting, each new model is trained to correct the errors of the previous one by fitting to its residuals. However, since we use the same training data for each boosting iteration, the new residuals are computed using predictions of old residuals that have already been influenced by the target values through previous models. This creates a feedback loop where the meta-learner progressively adjusts itself to fit the training data more closely and eventually "sees" the true target. The distribution of predictions is shifted toward the target values, and the model's ability to generalize is reduced. With unlimited labeled data at hand, this effect could have been diminished by using unseen data at every boosting round. In practice, however, that is rarely the case.

CatBoost handles both these challenges using similar strategies based on random permutations of the data. It generates multiple random permutations, each introducing a unique ordering, and processes the data sequentially in a way that prevents target leakage and biased residual calculations. To encode categorical features, it employs ordered target encoding, where the encoding happens sequentially within each permutation. When encoding a categorical feature for a current sample, only the target values from previous samples in that permutation are used, ensuring that the true target for the current sample is not observed. This method prevents the model from "seeing" future target information during feature encoding, thus reducing target leakage. The encodings from all permutations are averaged to produce the final encoding for each categorical feature.

Similarly, to avoid biased residuals during the boosting process, CatBoost uses ordered boosting. Each base learner is trained sequentially on subsets of the data. Specifically, the i^{th} model is trained using only the first $i-1$ observations in the permutation. To estimate the residual for the i^{th} observation, the model trained on observations 1 to $i-1$ is used. This ensures that each prediction is made without access to future target values, preventing overfitting due to target leakage in gradient estimation. By combining ordered target encoding with ordered boosting, CatBoost effectively enhances generalization and reduces the risk of overfitting. Furthermore, it often delivers state-of-the-art accuracy with default settings, requiring minimal parameter tuning (Prokhorenkova et al., 2017), which makes it both powerful and easy to use.

2.3.7 Shrinkage-Based Linear Regression

Classical linear regression remains a cornerstone within the field of ML, offering simplicity, interpretability, and computational efficiency. In its classical form, *ordinary least squares* (OLS) minimizes the squared error between observed and predicted values

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} \;=\; \underset{\boldsymbol{\beta}}{\text{argmin}}\; \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \;=\; (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}\;,$$

where $\mathbf{X} \in \mathbb{R}^{n\times p}$ is a design matrix, $\mathbf{y} \in \mathbb{R}^n$ is the response, and $\boldsymbol{\beta} \in \mathbb{R}^p$ contains the model parameters. This setup typically works well when the relationships between the response $\mathbf{y}$ and the features $\mathbf{X}$ are linear and the degree of multicollinearity is low. In high-dimensional settings, however, the problem becomes ill-posed and does not yield a unique solution. One way to combat that is through shrinkage-based methods, which penalize coefficient sizes to reduce variance and improve generalization. In this thesis, we apply two such models: RR (Hastie et al., 2009; James et al., 2023), used as a standalone model, and the *elastic net* (EN, Zou and Hastie, 2005), used as a linear base learner within the XGBoost framework.

Ridge Regression

RR is an extension of OLS that puts a penalty on the magnitude of model parameters, shrinking them towards zero. The model solves

$$\underset{\boldsymbol{\beta}}{\text{argmin}}\; \{\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2\;\}, \qquad \hat{\boldsymbol{\beta}}_{\text{ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}\;,$$

where $\lambda > 0$ controls the strength of the shrinkage (the intercept is typically left unpenalised). When parameters cannot grow too large in magnitude, the model becomes more robust and predictions for new data points are less sensitive to minor variations in the training data. Finding an optimal λ usually involves cross-validation over a grid of candidate values. The λ giving the lowest cross-validation error is then selected for the final model. As noted in Section 2.2.2, RR is well established within GP and used here as a fast and scalable alternative to both the animal model and the gradient boosted models.

Elastic Net as Base Learner

While gradient boosting models typically use decision trees, other models, such as linear models, splines, and even neural networks, can also serve as base learners. Since the XGBoost framework allows linear models, we include a variant that uses EN instead of decision trees, providing an alternative to tree-based boosting. EN extends RR by adding another penalty term to the model, encouraging sparsity in addition to shrinkage. While RR only shrinks coefficients, EN can drive some coefficients all the way to zero, effectively performing variable selection. The model solves

$$\hat{\boldsymbol{\beta}}_{\text{EN}} = \arg\min_{\boldsymbol{\beta}}\; \{\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \;+\; \lambda_1\|\boldsymbol{\beta}\|_1 \;+\; \lambda_2\|\boldsymbol{\beta}\|_2^2\},$$

with both λ_1 and λ_2 as tunable parameters controlling the penalties.

2.3.8 Regularization and Hyperparameters

In previous discussions, we mentioned regularization and overfitting without further elaboration. Overfitting in ML occurs when a model is too complex, fitting the training data so

closely that it captures noise and specific variations that do not generalize to unseen data. As a result, the loss decreases on training data but increases on unseen data (Mitchell, 1997; Tian and Zhang, 2022). Regularization is our way of combating overfitting by improving the generalization ability of a model (Tian and Zhang, 2022). The idea behind regularization is to add a constraint to the loss function that cannot be derived from the data itself, discouraging the model from fitting the training data too perfectly. The constraint reduces the complexity by removing or shrinking feature weights towards zero.

As an example, consider the regularization term used in Section 2.3.4, describing XGBoost,

$$\Omega(f^{(t)}) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \ ,$$

where γ controls the penalty for the number of leaf nodes T in the tree, and λ determines the penalty for the magnitude of the leaf weights w_j. A larger γ encourages smaller, simpler trees, while a larger λ leads to smaller weights, reducing model variance and improving generalization. Both γ and λ are hyperparameters. All the models previously discussed in Section 2.3, except OLS, rely on a set of hyperparameters – external settings for the models that are set prior to training and are not learned from the data. Hyperparameters control various aspects of the training procedure, such as learning rate, tree depth, and other regularization parameters. They can have a significant impact on model performance and often require careful tuning to obtain the best possible configuration. Such tuning leads to the concept of hyperparameter optimization, a critical process for ensuring the best possible performance.

Hyperparameter Optimization

Hyperparameter optimization aims to fine-tune model settings to achieve optimal performance. The procedure usually involves defining a hyperparameter space (or grid of possible parameter values). Techniques like *grid search* and *random search* can then systematically or randomly explore the hyperparameter space (Bergstra and Bengio, 2012). However, these methods may not be the most efficient, especially when dealing with computationally expensive models and large parameter spaces. Evaluating every possible combination of hyperparameters is often impractical. To address this challenge, we employ *Bayesian optimization*, which has been shown to outperform the two other methods in hyperparameter tuning when the number of allowed evaluations is constrained (Bergstra et al., 2011). Bayesian optimization is an effective strategy for global optimization of computationally heavy black-box derivative-free functions (Frazier, 2018; Garnett, 2023) – functions that are expensive to evaluate and lack analytical gradients, where we only observe input and output.

Bayesian optimization leverages principles from Bayesian statistics to optimize objective functions. In the context of hyperparameter optimization, the objective function represents the model's performance metric (*e.g.*, sum of squared errors) as a function of the hyperparameters. The process begins with defining a prior belief about the objective function. This prior model predicts the objective function's behavior based on the hyperparameters. As we evaluate the objective function at different hyperparameter settings, we update the prior belief to form a posterior model. This updating process is guided by Bayes' theorem, which in Bayesian statistics allows us to update our beliefs based on new evidence. Bayes' theorem is formalized as

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \ ,$$

where $P(\theta|D)$ is the posterior probability of the hyperparameters given the data D, $P(D|\theta)$ is the probability of observing data D given hyperparameters θ, $P(\theta)$ is the prior belief of the hyperparameters before observing data, and $P(D)$ is the marginal probability of the data, serving as a normalizing constant.

At each iteration, an *acquisition function* is used to select the next set of hyperparameters to be evaluated. The acquisition function balances *exploration* (searching new areas of the hyperparameter space) and *exploitation* (refining known good areas) by considering both the predicted performance and the uncertainty in the prior model. By intelligently selecting hyperparameter settings to evaluate, Bayesian optimization can find optimal or near-optimal hyperparameters with fewer evaluations compared to grid search or random search (an illustration of the three concepts is shown in Figure 3), making it especially valuable when training models is computationally expensive. A comprehensive overview of Bayesian optimization can be found in Frazier (2018).

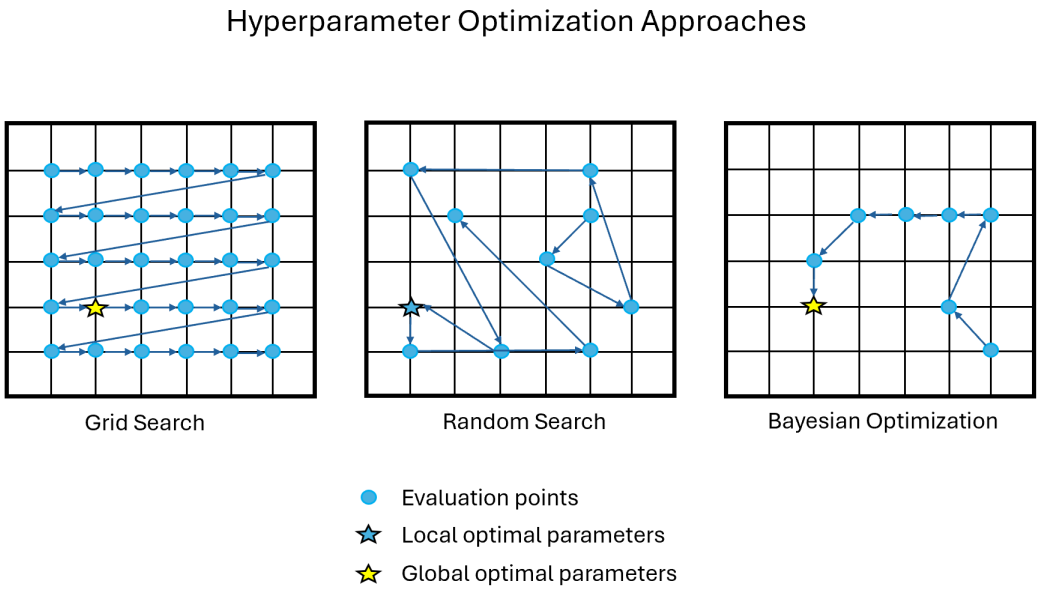


Figure 3: Illustrations of Grid Search, Random Search, and Bayesian optimization for hyperparameter optimization. The figure demonstrates how each strategy explores a grid of possible hyperparameter values. Grid Search exhaustively evaluates all combinations, while Random Search samples configurations at random – potentially missing good regions without sufficient trials. Bayesian optimization, by contrast, leverages past trials to guide the search and is expected to find an optimal or near-optimal solution in fewer iterations.

2.4 Dimension Reduction with t-SNE

As part of an exploratory data analysis, dimension reduction techniques can be used to visualize the dataset, possibly revealing underlying structure. Such visualizations are particularly interesting when studying wild animal populations, where subpopulations may

be subject to diverse environments and have limited interbreeding. Identifying structure visually can help us understand how genetically distinct the subpopulations are, and whether such patterns might bias or limit predictive modeling. If individuals cluster by subpopulation, it suggests that a model trained on one group may not generalize well to another – consistent with findings by Wientjes et al. (2015); Goddard et al. (2016); McGaugh et al. (2021), who emphasize the importance of relatedness between training and test populations. Conversely, if subpopulations are genetically intertwined, we would expect better generalization. In this thesis, we use dimension reduction to project high-dimensional SNP data into a three-dimensional space for visual inspection, helping us assess population structure.

A thorough analysis of the same data was conducted by Singsaas (2024), using both *principal component analysis* (PCA) and *t-distributed stochastic neighbor embedding* (t-SNE), the latter explained by Van Der Maaten and Hinton (2008). The output obtained from t-SNE relies on hyperparameters and is non-deterministic, meaning randomness influences the results. However, according to Singsaas (2024), the observed patterns were similar across different sets of parameters tested. Thus, in this thesis, we take a simpler approach, presenting only a single result using t-SNE, with the purpose of highlighting potential clusters. While PCA assumes linear relationships in the data, t-SNE has no such restriction, making it a powerful tool for dimension reduction. The idea behind t-SNE is to project high-dimensional data to lower dimensions, preserving most of the local structure and maintaining relative distance between similar data points.

A detailed explanation of the algorithm and the math behind it is beyond the scope of this thesis, but the core concepts are worth understanding. Suppose we have the data points $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ in p-dimensional space. Our goal is to project these to $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$ in 2 or 3 dimensions. The first question is: given x_i, what is the probability that x_j will be picked as its neighbor? This probability can be calculated using the following metric (Van Der Maaten and Hinton, 2008)

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k\neq l}\exp(-\|x_k - x_l\|^2/2\sigma^2)} \ , \tag{8}$$

where $p_{j|i}$ is the probability that x_i selects x_j as its neighbor (with $p_{i|i} = 0$, and the distance metric $\|\cdot\|$ choosen by the user). Since this is a division of Gaussian probability density functions, we can read $p_{j|i}$ as the "*conditional probability that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i*" (Van Der Maaten and Hinton, 2008). The overall probability p_{ij} that x_i and x_j are grouped together is

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \ .$$

Similarly, we calculate the probability for the corresponding lower-dimensional data points y_i and y_j as

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}} \ .$$

However, in lower dimensions, the distribution is a 1-degree-of-freedom t-distribution (or Cauchy distribution) rather than a Gaussian one. To preserve the relationships from the

higher dimensions to the low-dimensional space, we need p_{ij} and q_{ij} to be as close as possible. Since both represent probability density functions, we can use the *Kullback-Leibler* (KL) divergence to measure the difference between them. KL divergence is a type of statistical distance that measures how one probability density function differs from another. In t-SNE, it serves as the cost function and is minimized using *gradient descent*. With P and Q representing the joint densities of the high-dimensional probabilities and the low-dimensional probabilities, respectively, we minimize the function

$$KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \ .$$

The key hyperparameters for t-SNE are perplexity, learning rate, and number of iterations. The perplexity parameter controls the variance, σ^2, of the Gaussian distribution in high-dimensional space (equation (8)) and can be thought of as the effective number of neighbors per data point. The learning rate and number of iterations govern the optimization process, determining the step size and how long the algorithm runs for.

t-SNE is excellent at revealing local patterns but often distorts global structure, meaning cluster distances may not reflect true relationships in high-dimensional space. Also, larger clusters don't necessarily indicate more genetic diversity. However, overlapping clusters indicate gene flow between the populations represented. These properties are important to remember when interpreting t-SNE visualizations.

2.5 Subset Selection

GP models may rely on large-scale SNP data, involving hundreds of thousands of markers. High dimensionality enables models to potentially capture complex genetic signals, but also poses significant computational challenges. Training predictive models with large feature sets is time-consuming and resource-intensive, especially when combined with cross-validation and hyperparameter optimization. Beyond practical considerations, an interesting question in the context of GP is how many SNPs are actually necessary to capture the genetic signal relevant for a given phenotype. If a relatively small subset can provide comparable predictive power to the full marker set, this has implications not only for computational efficiency but also for interpretability and our understanding of the genetic architecture of the trait. These are compelling reasons to experiment with feature subsets or other forms of dimensionality reduction. There is also a theoretical motivation rooted in the *bias-variance tradeoff*, which describes how model performance can change as more features are included. In models susceptible to overfitting, such as tree-based methods, performance may initially improve as added features reduce bias, but can eventually plateau or decline as variance increases and noise is incorporated (Hughes, 1968; Trunk, 1979; James et al., 2023). This behavior depends on both the nature of the data and the modeling approach. If such a tradeoff arises in our setup, then intermediate subset sizes may actually outperform the full SNP set, reducing computational demands and improving predictive accuracy.

In Wold (2024), we experimented with gradient boosting algorithms on randomly sampled SNP subsets of varying sizes, up to 25,000 markers. This was partly inspired by findings from Bérénos et al. (2014) and Singsaas (2024), who observed diminishing performance returns beyond 18,000 - 20,000 SNPs. Our results showed a similarly rapid plateau in model accuracy; in some cases, models trained on less than 1,000 SNPs performed comparably

to those trained on 14,000 or more. Based on those early results, we selected a subset of 15,000 SNPs for the main prediction task. However, in hindsight, this choice may have been premature. The upper limit of 25,000 SNPs left open the question of whether larger subsets might still yield improvements. To address this question, we extended the original experiment to include a broader range of subset sizes. This new experiment aims to clarify whether larger SNP sets lead to improved performance, and how the relationship between subset size and accuracy differs across models and phenotypes.

2.6 Feature Extraction by PCA and Autoencoders

2.6.1 Introduction and Motivation

To expand from the subset selection in Section 2.5, we examined *feature extraction* techniques that compress the SNPs while aiming to preserve most of the genetic signal. Feature extraction creates new composite variables from the original markers, distilling the data into a lower-dimensional representation that – ideally – retains only the most relevant information. Besides serving as a tool for two- or three-dimensional visualization, a good lower-dimensional representation can also be a powerful input to supervised learning algorithms, speeding up training and sometimes improving predictive accuracy (Bengio et al., 2013; James et al., 2023). We included two approaches for dimensionality reduction. The first was the classical linear method, PCA. The second was a non-linear, neural-network-based method known as an *autoencoder*. An autoencoder is trained to compress the input data into a lower-dimensional representation (encoding) and, from there, reconstruct it. By the *universal approximation theorem* (Cybenko, 1989), even a shallow neural network can approximate any continuous function, so autoencoders are well suited to capture non-linear structure in the data. Although Singsaas (2024) applied neural networks directly for GP on this sparrow dataset, using them for representation learning remains unexplored. As the reader is assumed to be familiar with PCA, the remainder of this section reviews the essentials of neural networks and explains how they are configured as autoencoders, starting with a simple single-layer neural network to set the stage. The following description of a single-layer neural network is borrowed from James et al. (2023).

2.6.2 Single-Layer Neural Network

A simple single-layer neural network is shown schematically in Figure 4. It consists of an input layer taking an input of p features $X = (X_1, X_2, ..., X_p)$, a hidden layer of K units, each applying a non-linear *activation function* g specified in advance, and an output layer that combines these hidden units' outputs, for example for regression or classification. The edges indicate that each input node feeds into each of the K hidden units and that the output from the hidden units feeds into the output layer. The output, $f(X)$, is constructed in multiple steps. First the K *activations* A_k, $k = 1, ..., K$ in the hidden layer are computed from the input,

$$A_k = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j) \qquad k = 1, .., K \ ,$$

where w_{k0} is a bias term and w_{kj} are the weights assigned to each input feature. Further, the activations feed into the output layer. For a regression task, we can form the final

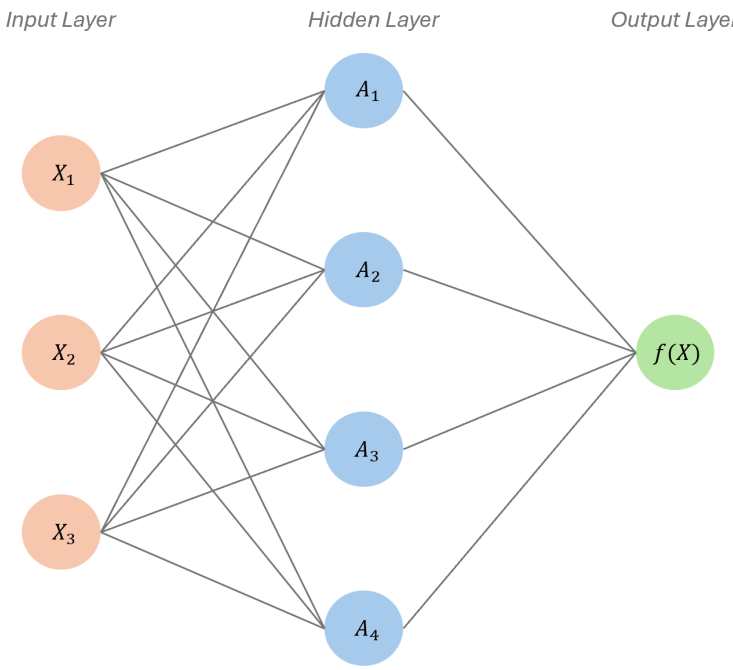


Figure 4: Graphical representation of a single-layer neural network. The hidden layer computes activations A_k, $k = 1, ..., K$, which are non-linear transformations of the input X. The output layer then uses these activations as inputs to produce the final prediction $f(X)$.

output $f(X)$ as a linear combination of these activations, allowing it to take any value in $\mathbb{R}$ by

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j) \ . \tag{9}$$

Here, β_0, β_k are weights from the hidden unit k to the output node. For classification, one might apply a sigmoid at the output instead

$$f(X) = \sigma(\beta_0 + \sum_{k=1}^{K} \beta_k A_k), \qquad \text{with} \quad \sigma(x) = \frac{1}{1 + e^{-x}} \ .$$

In both cases, the activation functions are predetermined by a user, while the w's and the β's are the learnable parameters.

2.6.3 Fitting by Gradient Descent and Backpropagation

Fitting a neural network means estimating the weights in (9) by minimizing a loss function that measures how closely $f(\mathbf{x}_i)$ resembles the true response y_i. Having a quantitative response, one typically uses squared-error loss, so the parameters are optimized to minimize

$$R(\theta) = \frac{1}{2} \sum_{i=1}^{N} (y_i - f_\theta(\mathbf{x}_i))^2 \ ,$$

where $\mathbf{x}_i \in \mathbb{R}^p$ is the i^{th} input, y_i is the i^{th} target, and f_θ highlights the networks's dependence on the trainable parameters, which we have bundled into to vector θ. We want to find θ that minimizes $R(\theta)$. A common approach is gradient descent, where we iteratively update by

$$\theta^{t+1} = \theta^t - \rho \nabla R(\theta^t) \ , \tag{10}$$

with learning rate ρ. The critical step is computing the gradient $\nabla R(\theta^t)$. This is done by *backpropagation*, which utilizes the chain rule to propagate the residual term $y_i - f_\theta(\mathbf{x}_i)$ backward through the network layer by layer via partial derivatives. At each layer, a fraction of the residual is assigned to each neuron and its associated parameters based on how much they influence the output. When the partial derivatives for all the weights are computed, we update by (10). Since $R(\theta) = \sum_i^N R_i(\theta)$ is a sum, the gradient is also a sum. Let's consider one term in the sum,

$$R_i(\theta) = \frac{1}{2}\left(y_i - f_\theta(\mathbf{x}_i)\right)^2 = \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\right)\right)^2 \ .$$

For ease of notation we define $z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}$. Then $A_k = g(z_{ik})$. The partial derivative with respect to β_k is

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = \frac{\partial R_i(\theta)}{\partial f_\theta(\mathbf{x}_i)} \cdot \frac{\partial f_\theta(\mathbf{x}_i)}{\partial \beta_k} = -(y_i - f_\theta(\mathbf{x}_i)) \cdot g(z_{ik}) \ .$$

Similarly, for w_{kj}

$$\frac{\partial R_i(\theta)}{\partial w_{kj}} = \frac{\partial R_i(\theta)}{\partial f_\theta(\mathbf{x}_i)} \cdot \frac{\partial f_\theta(\mathbf{x}_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}} = -(y_i - f_\theta(\mathbf{x}_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot xij \ .$$

Notice how a "chain" of partial derivatives emerges, each passing a fraction of the residual back through the network. These types of layers, with linear combinations, non-linear activation functions, and backpropagation for weight updates, are also used in larger and deeper architectures, including autoencoders.

2.6.4 The Autoencoder Concept

An autoencoder (Figure 5) is a neural network designed to learn a so-called identity mapping – that is, to reproduce its input at the output layer. The input goes through a bottleneck layer of reduced dimensionality, forcing the network to produce a compressed representation of the input. Early adaptations of autoencoders were proposed already in the 1980s (Rumelhart et al., 1986; Bourlard and Kamp, 1988; Kramer, 1991), and they remain popular for tasks like file compression, anomaly detection, data generation, and denoising (Goodfellow et al., 2016). The autoencoder has two main entities, the *encoder* and the *decoder*. The encoder is a function $\mathbf{z} = E(\mathbf{x})$ mapping the input $\mathbf{x} \in \mathbb{R}^p$ to a *latent representation* $\mathbf{z} \in \mathbb{R}^k$ (with $k < p$). The decoder is a function $\hat{\mathbf{x}} = D(\mathbf{z})$ that aims to reconstruct $\mathbf{x}$ from $\mathbf{z}$, such that $D(E(\mathbf{x})) = \mathbf{x}$. Training the autoencoder, we use a loss function that measures the reconstruction error between $\hat{\mathbf{x}}$ and $\mathbf{x}$. Minimizing this error pushes the network to learn a meaningful compression of the input. Because

$k < p$, the network cannot simply learn the trivial identity mapping, it must compress relevant information into the latent layer. After training, we often discard the decoder and use the encoder's latent output $\mathbf{z}$ as a new, reduced set of features for downstream tasks, as we aim to do for our SNP data in this thesis. However, the network we will use is not only comprised of *dense layers*, as those depicted in Figure 4 and Figure 5. In practice, neural networks, including autoencoders, can be built from various components depending on their use. The one used in this thesis is comprised of five main layer types:

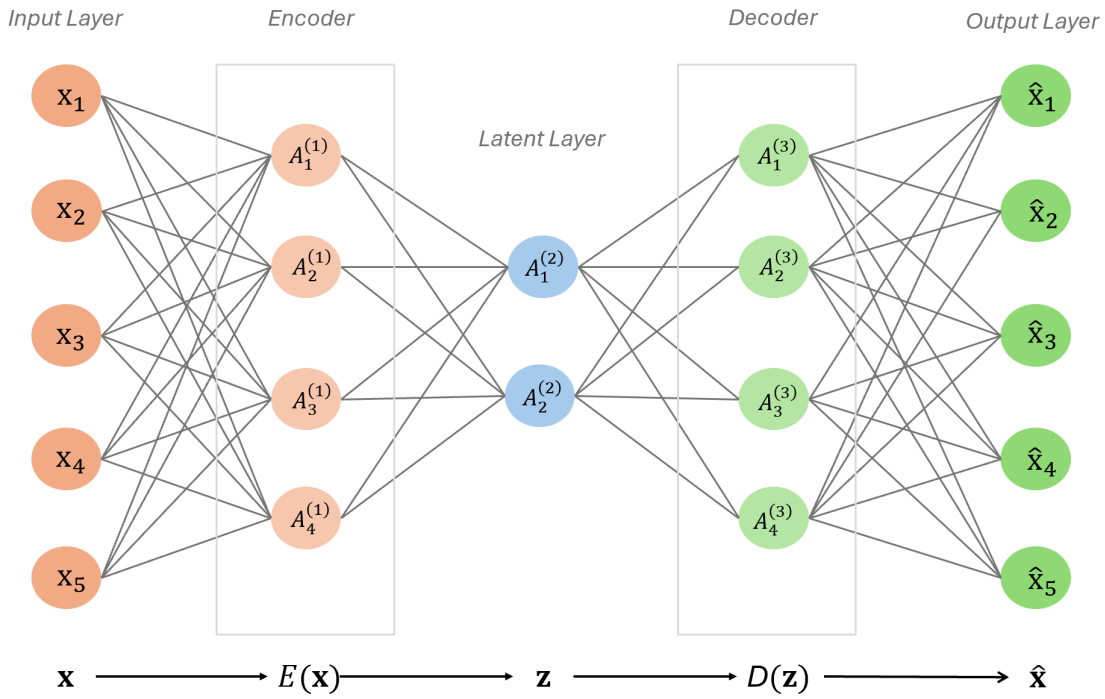


Figure 5: Graphical representation of a simple autoencoder used for dimension reduction. The network consists of an input layer, three hidden layers, and an output layer. The encoder maps the high-dimensional input $\mathbf{x} \in \mathbb{R}^p$ to a compressed latent representation $\mathbf{z} \in \mathbb{R}^k$, $(k < p)$. The middle hidden layer, the latent layer, captures this compressed representation. Then the decoder attempts to reconstruct the original input from the latent representation, forcing the network to learn an informative representation.

convolutional layers, pooling layers, dense layers, upsampling layers, and normalization layers. In addition, we make use of *residual blocks*, which have been shown to make optimization of deep neural networks easier (He et al., 2015). The following is a more detailed description of the layers and components used in our network.

2.6.5 Layer Types

Dense Layers

In a dense layer, every neuron is connected to every output of the previous layer (Figure 4). For an input vector $\mathbf{x}$ of size p, a single dense layer neuron computes the weighted sum $w_0 + w_1x_1 + w_2x_2 + ... + w_px_p$, then applies an activation function to that sum, often ReLu or sigmoid. Dense layers are flexible because each neuron can combine all inputs. A downside with dense layers is the computational cost when p grows because of the rapidly

increasing number of parameters to compute.

Convolutional Layers

A *convolutional layer* (Figure 6) slides a small *filter* (or kernel) across the input, computing the weighted sum between the filter's weights and the local patch of the input. In image processing, this helps detect edges and patterns. For genomic data, a 1D convolution can identify meaningful subsequences. Because the same filter is applied across the entire input, convolutional layers are typically more parameter-efficient than dense layers. The size of the output is determined by how the filter is applied – specifically, its starting position, *stride* (the step size at each move), and the use of *padding* to maintain shape.

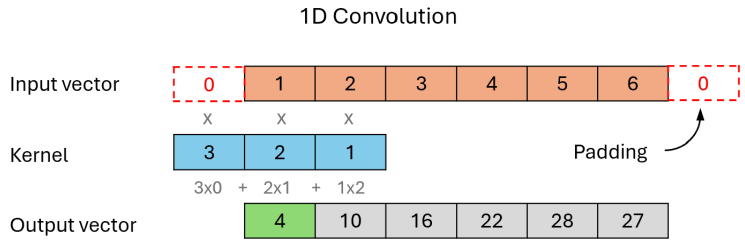


Figure 6: Illustration of a 1D convolutional layer. By padding with zeros at the beginning and end of the input vector, the output becomes the same shape as the original input.

Pooling Layers

Pooling layers are used to downsample the output from convolutional layers by combining nearby values into a single number. Common strategies are *max pooling* (Figure 7), which selects the maximum value within each region, and *average pooling*, which computes the mean. This aggregation shrinks the data, cutting computational costs and helps generalize. The chosen *pool size* controls how many consecutive elements you aggregate (e.g., 2 or 3 data points per pool), while the stride determines how far you move the pooling window after each aggregation step, which directly affects the compression factor. Together, they set both how many values go into each pool and how aggressively you reduce the spatial dimension.

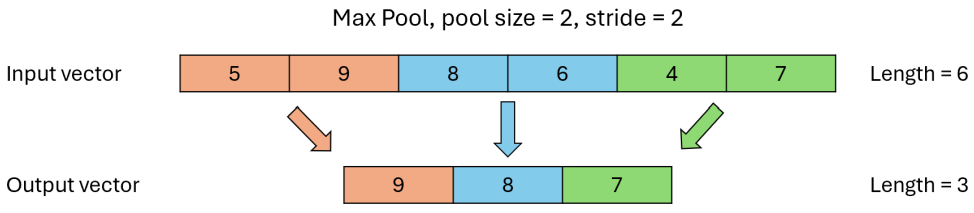


Figure 7: Illustration of a pooling layer, downsampling the input by choosing only the maximum value from each pool. This layer uses both pool size and stride equal to 2, which means the output will be exactly half the length of the input.

Upsampling Layers

An upsampling layer is used in the decoder part of the autoencoder to increase dimensionality after the bottleneck, reversing the reduction performed by the pooling. While pooling layers reduce the input size, upsampling layers expand it. There are various methods for upsampling; here, we use the *nearest neighbor* approach, simply duplicating values to increase the size. For example, given the reduced vector $[9, 8, 7]$ from Figure 7, the upsampled vector would be $[9, 9, 8, 8, 7, 7]$.

Normalization Layers

Normalization or *batch normalization* is not a traditional layer but rather an operation inserted between layers. During training, the distribution of a layer's inputs can shift as upstream parameters change, a phenomenon called *internal covariate shift* (Ioffe and Szegedy, 2015). This shift complicates optimization, necessitating lower learning rates and careful parameter initialization, particularly in deeper networks with non-linearities. Normalizing within each batch of training data sent through the network, we stabilize training with respect to hyperparameters, allow for faster convergence, and help regularization (Ioffe and Szegedy, 2015).

Residual Block

In a standard layer, an input $\mathbf{x}$ is transformed through a series of operations to produce an output y. A residual block (Figure 8), by contrast, includes a skip connection that allows the input $\mathbf{x}$ to bypass the intermediate layers and be added directly to the output. These blocks learn a residual function with respect to the input, rather than mapping the input directly to the output. If the full mapping is denoted by $H(\mathbf{x})$, the residual block only learns $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$, such that the output becomes $F(\mathbf{x}) + \mathbf{x}$. This design allows the block to easily revert to the identity function if needed. If the learned transformation $F(\mathbf{x})$ is not useful, the model can simply pass through the original input.

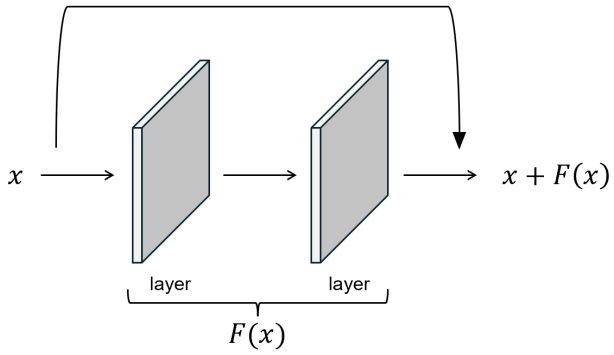


Figure 8: Structure of a residual block with two intermediate layers forming the residual function $F(\mathbf{x})$.

To summarize this section, we've covered the fundamentals of neural networks, including how dense layers learn parameters via backpropagation and gradient descent. We then showed how autoencoders leverage this machinery for dimensionality reduction by reconstructing their input after passing it through a narrow bottleneck. Finally, we briefly introduced the layer types and components used in the network employed in this thesis.

In Section 3.7, we will dig deeper into our specific autoencoder, its implementation, and its applications.

3 Methods

3.1 The Sparrow Data

The data used in this thesis comes from a meta-population of Norwegian house sparrows (*Passer Domesticus*) distributed across 16 islands along the Helgeland coast. It contains repeated measurements of phenotypic values for three physical traits – body mass, tarsus length, and wing length – along with allele counts for 182,854 SNPs and additional environmental features. The environmental features are displayed in Table 1. The measurements have been collected annually from 1993 to 2017. In total, data from 1,918 unique individuals are available. The distribution of individuals across the islands is presented in the table in Figure 9 (left).

Table 1: Environmental variables used in modeling.

Variable	**Meaning**
Sex	Gender of individual
FGRM	Inbreeding coefficient of individual
Month	Month of measurement
Age	Age of individual
Outer	Proportion of genetic material from *outer* group
Other	Proportion of genetic material from *other* group
Island_current	Measurement made at this island
Hatchisland	Island where individual hatched
Hatchyear	Year of hatching

The metapopulation spans two major habitat types: the *inner islands*, closer to the mainland, where sparrows breed in colonies on farms; and the *outer islands*, farther from the mainland, where sparrows face harsher environmental conditions and lack access to farms (Ranke et al., 2018). This split is highlighted in Figure 9 (right). While the metapopulation can be treated as a whole, it can also be examined at an island level, treating each island as its own population. Some islands are more isolated than others, but there is a known genetic flow between all of them (Ranke et al., 2018), as illustrated in Figure 12.

Island	Sample size
Nesøy	103
Sundøy	2
Myken	37
Træna	100
Selvær	108
Gjerøy	418
Hestmannøy	719
Indre Kvarøy	247
Ytre Kvarøy	2
Selsøyvik	4
Lurøy	1
Onøy	9
Lovund	6
Sleneset	6
Rødøy	3
Aldra	153

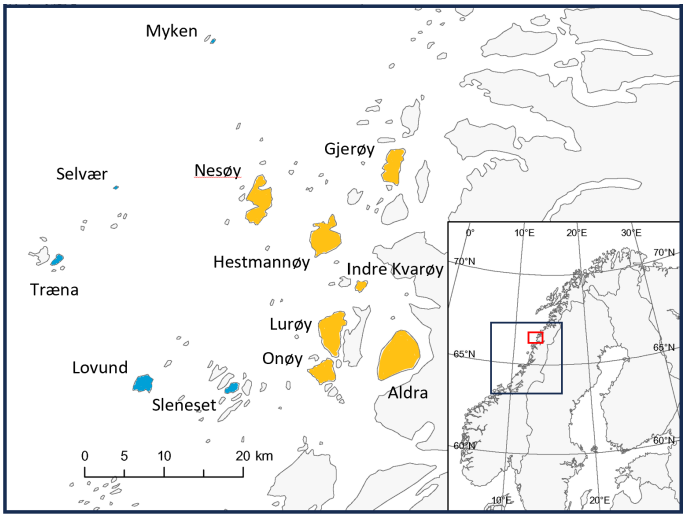


Figure 9: Left: Table showing the names of the islands and the corresponding number of sampled individuals per island. Right: Map illustrating the island system, with islands of larger sample sizes labeled and color-coded. The inner islands are marked in orange, while the outer islands are marked in blue.

3.2 Animal Model and GBLUP by INLA

To assess the relative performance of other methods, we fitted a genomic version of the animal model using the Integrated Nested Laplace Approximation (INLA, Martino and Riebler, 2019). This model represents the current state-of-the-art for GP in quantitative genetics and served as a reference for the ML models evaluated later. Our goal with this approach was to predict breeding values, which reflect the additive genetic contribution of each individual to a trait. To separate them from other non-additive or individual-specific effects, we explicitly included both the breeding value a_i and an individual-specific ID-effect, $w_{\text{ID},i}$. The ID-effect captures permanent environmental influence or genetic factors not explained by the breeding values and the fixed effects. The model can be written as follows

$$\begin{aligned} y_{ij} = \mu + \beta_1 \text{sex}_i + \beta_2 \text{FGRM}_i + \beta_3 \text{age}_i + \beta_4 \text{month}_i + \beta_5 \text{outer}_i + \beta_6 \text{other}_i + \\ w_{\text{island\_current},ij} + w_{\text{hatchyear},i} + w_{\text{ID},i} + a_i + \varepsilon_{ij} \ , \end{aligned} \tag{11}$$

where y_{ij} is the j^{th} measurement of individual i, μ is the overall intercept, β_k represent fixed effects, while $w_{\text{island\_current}}$, $w_{\text{hatchyear}}$, and w_{ID} are the random effects. The breeding value a_i is assumed to follow a multivariate normal distribution with covariance structure proportional to the GRM, while ε_{ij} is *i.i.d.* Gaussian noise. In matrix form, the model is written more compactly as

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{X}\boldsymbol{\beta} + \mathbf{W}\mathbf{d} + \boldsymbol{a} + \boldsymbol{\varepsilon} \ , \tag{12}$$

with $\mathbf{X}$ and $\mathbf{W}$ being design matrices for the fixed effects $\boldsymbol{\beta}$ and random effects $\mathbf{d}$, respectively, and the random effects $\mathbf{d}$ are distributed $\mathbf{d} \sim N(\mathbf{0}, \mathbf{R})$. The vector $\boldsymbol{a} \sim N(\mathbf{0}, V_A\mathbf{G})$

holds the individual breeding values, with $\mathbf{G}$ being the GRM and V_A the additive genetic variance. Finally, $\boldsymbol{\varepsilon} \sim N(\mathbf{0}, \sigma^2\mathbf{I})$ is the residual error term. Combining $\boldsymbol{\mu}$ and $\boldsymbol{\beta}$ into $\boldsymbol{\beta^*}$ with corresponding design matrix $\tilde{\mathbf{X}}$, and $\boldsymbol{a}$ and $\mathbf{d}$ into a single vector $\boldsymbol{\gamma}$ with a block diagonal design matrix $\mathbf{U}$, we can write (12) even more compact as

$$\mathbf{y} = \tilde{\mathbf{X}}\boldsymbol{\beta}^* + \mathbf{U}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim N(\mathbf{0}, \sigma^2\mathbf{I}), \qquad \boldsymbol{\gamma} = \begin{pmatrix} \boldsymbol{a} \\ \mathbf{d} \end{pmatrix} \sim N\left(\mathbf{0}, \begin{pmatrix} V_A\mathbf{G} & 0 \\ 0 & \mathbf{R} \end{pmatrix}\right) = N(\mathbf{0}, \mathbf{C}) \ .$$

Assuming $\boldsymbol{\gamma}$ and $\boldsymbol{\varepsilon}$ are independent, the distribution of $\mathbf{y}$ becomes

$$\mathbf{y} \sim (\tilde{\mathbf{X}}\boldsymbol{\beta^*}, \mathbf{V}), \qquad \text{where } \ \mathbf{V} = \mathbf{U}\mathbf{C}\mathbf{U}^\mathbf{T} + \mathbf{R} \ . \tag{13}$$

From equation (13), it can be shown that the best linear unbiased estimator (BLUE) of the fixed effects is given by

$$\hat{\boldsymbol{\beta}}^* = (\tilde{\mathbf{X}}^\mathbf{T}\mathbf{V}^{-1}\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^\mathbf{T}\mathbf{V}^{-1}\mathbf{y} \ ,$$

(Henderson, 1975, 1984), and the best linear unbiased predictor (BLUP) of the random effects, including the breeding values

$$\hat{\boldsymbol{\gamma}} = \mathbf{C}\mathbf{U}^\mathbf{T}\mathbf{V}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\hat{\boldsymbol{\beta}}^*) \ .$$

However, since we used the GRM and not a pedigree-based relationship matrix, we call it the genomic BLUP, or GBLUP. To fit this model in a Bayesian framework, we used INLA, which is designed for *Latent Gaussian Models* (LGMs), a class of models which includes the animal model as a special case. While the detailed workings of INLA are beyond this thesis, it provides a fast and accurate alternative to traditional MCMC for approximating posterior distributions. Using INLA, we obtained approximate marginal posterior distributions for the model parameters. The estimated breeding values were then extracted by taking the posterior means of the distributions corresponding to the genetic effects. From this point onward, the estimated breeding values are referred to as *genomically estimated breeding values* (GEBVs).

3.3 The Two-Step Procedure

In GP, we aim to model the genetic contribution to each individual's phenotype. However, phenotypes also reflect environmental variation, which can confound predictions. A common strategy to address this issue is a two-step approach, separating as much of the environmental signal as possible from the genetic effects before applying predictive models, (Ashraf et al., 2021; Hunter et al., 2022). The idea is to create an adjusted phenotype that captures the remaining variation after accounting for fixed and random environmental effects. This new phenotype should contain the full genetic signal plus any remaining individual-specific permanent environmental effects. In contrast to the animal model (Section 3.2), we do not estimate breeding values directly in a single unified model that combines fixed effects, random effects, and a genomic relationship matrix within a Bayesian framework. Instead, we take a two-step approach: we first isolate individual-level effects using a linear mixed model (LMM), fitted with frequentist methods, and then use these estimated effects as adjusted phenotypes to be predicted in a second step using genomic data and machine learning models. Following the approach in Wold (2024), we fitted the model

$$y_{ij} = \mu + \beta_1 \text{sex}_i + \beta_2 \text{FGRM}_i + \beta_3 \text{age}_i + \beta_4 \text{month}_i + \beta_5 \text{outer}_i + \beta_6 \text{other}_i + w_{\text{island\_current},ij} + w_{\text{hatchyear},i} + w^*_{\text{ID},i} + \varepsilon_{ij} \ . \tag{14}$$

Equation (14) resembles equation (11), but there is a key difference: the breeding value a_i and the individual ID-effect $w_{\text{ID},i}$ from the animal model are here replaced by a single individual-specific effect $w^*_{\text{ID},i}$. This term is intended to absorb both the additive genetic effects, any non-additive genetic effects, and permanent environmental effects not accounted for by the environmental variables. Importantly, these effects are modeled as independent, without any correlation structure from a GRM.

The model is implemented in *R* using the `lme4` package. In matrix form, we can write it as

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{X}\boldsymbol{\beta} + \mathbf{W}\mathbf{d} + \boldsymbol{w}^*_{\text{ID}} + \boldsymbol{\varepsilon} \ .$$

We extracted the estimated individual random effects $\hat{\boldsymbol{w}}^*_{\text{ID}}$ as adjusted phenotypes

$$\mathbf{y}^* = \hat{\boldsymbol{w}}^*_{\text{ID}} \ .$$

In the second step, these were then used as response variables, where we fitted gradient boosted models or RR to predict $\mathbf{y}^*$ from the SNP data $\mathbf{Z}$. That is, we modelled

$$\hat{\mathbf{y}}^* = \hat{f}(\mathbf{Z}) \ .$$

The second step, including hyperparameter tuning and evaluation, was performed in *Python*.

3.4 Evaluating Model Performance

We assessed model performance by calculating the Pearson correlation between the predicted value and the mean phenotype of each individual. For the ML models, the predicted values are $\hat{\mathbf{y}}^*$, while for the animal model, they are the predicted breeding values $\hat{\boldsymbol{a}}$. For this calculation, we thus had to compute the mean phenotype for each individual. Since each individual may have one or several measurements, we calculated the mean phenotype by averaging all the available measurements in the following way

$$\bar{y}_j = \frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} \mathbf{y}_i \ ,$$

where $\mathcal{I}_j$ is the set of indices corresponding to individual j, $j \in \{1, \ldots, n\}$, n being the number of unique individuals. The overall model accuracy is then

$$\text{Corr}(\boldsymbol{v}, \bar{\mathbf{y}}), \qquad \bar{\mathbf{y}} = \{\bar{y}_j\}_{j=1}^n \ ,$$

ranging from 0 to 1, where $\boldsymbol{v}$ represents the predicted values from an arbitrary model and the sample Pearson correlation between two vectors $\{x_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$ with means $\bar{x}$ and $\bar{y}$ is given as

$$\text{Corr}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \ .$$

3.5 Fitting and Evaluating the ML Models

As described in Section 3.3, our goal is to model the adjusted phenotypes $\mathbf{y}^*$ using a function f estimated via gradient boosting or RR. This section outlines the methodology used to obtain an estimate $\hat{f}$ of f and evaluate the performance of each model.

We consider two classes of models: gradient boosted frameworks and RR. Our exact approach towards them differs slightly, but the core idea is the same – rather than relying on a single accuracy estimate, we aim to obtain a distribution of accuracy measurements. This allows us to evaluate how sensitive the models are to changes in training and test data and ensures representative results for all models. To achieve this, we apply 10-fold *cross-validation* (CV), where the data is randomly partitioned into ten subsets, with each subset serving once as the test set while the remaining folds are used for training. The same data splits are also used when evaluating the animal model, ensuring comparability.

In some experiments using CatBoost, we use default parameters without any tuning. However, we also perform more extensive experiments involving hyperparameter optimization for both RR and the gradient boosted models. For these, we use *nested cross-validation*. In this setup, one fold is held out as a test set, while a second layer of cross-validation is used within the training data to identify the best hyperparameters. Once the best configuration is found, a final model is trained on the full inner training set and evaluated on the outer test fold. This process is repeated across all folds. Model performance is assessed using the Pearson correlation coefficient for each fold.

Given that gradient boosting models are computationally expensive and involve many hyperparameters, we use a Bayesian optimization algorithm to search the hyperparameter space efficiently (Section 2.3.8). The number of optimization trials per fold is set to 20. For RR, which is computationally cheaper and only has one hyperparameter, we use a simple grid search instead – defining an array of candidate values and evaluating every one.

The nested CV procedure is outlined in Algorithm 3. The gradient boosted models – XGBoost, LightGBM, and CatBoost – are implemented via the `xgboost`, `lightgbm`, and `catboost` Python packages, respectively. Models using elastic net base learners are also fit using the `xgboost` library, while RR is implemented using `scikit-learn`.

Algorithm 3 Nested Cross-Validation

1: Split data randomly into folds
2: Set N = Number of folds
3: **for** $i = 1, 2, \ldots, N$ **do**
4: Test set = fold_i
5: Train set = $\{\text{fold}_j | j = 1, \ldots, N, j \neq i\}$
6: Run CV with hyperparameter optimization on train set
7: Refit model with best parameters to the whole train set
8: Test on test set
9: **end for**
10: Output: Array of Pearson correlation from each test set

Several optimization algorithms utilize the Bayesian framework. In this thesis, we employ the Tree-structured Parzen Estimator Approach (TPE, Bergstra et al., 2011), applied through Optuna (Akiba et al., 2019), a Python-based hyperparameter optimization framework. Hyperparameter spaces for the gradient boosted models are listed in Appendix A,

and the optimal regularization parameters for RR are shown in Appendix C.

3.6 Subset Selection

To investigate how predictive performance varies with the number of SNPs used, we conducted a subset selection experiment across a range of SNP subset sizes. Specifically, we evaluated subsets from the size of 10,000 up to the full 180,000 in increments of 10,000. For each subset size, we randomly sampled the specified number of SNPs from the full marker set without replacement. The analysis was performed using two modeling approaches, CatBoost and RR. CatBoost's ability to perform well without tuning made it a convenient choice for this exploratory analysis. It allowed us to focus on broader trends without investing heavily in model optimization at each subset size. Conversely, RR served as a fast and simple linear approach, providing a valuable contrast to the non-linear gradient boosting method.

For each SNP subset size and each model type, we performed 10-fold cross-validation to estimate predictive accuracy. This process was repeated across all subset sizes and allowed us to quantify how model performance changes as the number of input features increases. The goal of this experiment was not to optimize model parameters for the best possible predictions, but to identify systematic patterns in performance as a function of SNP subset size, and to explore how these patterns may differ between phenotypes. By analyzing the prediction accuracy, we could also determine whether performance improvements continue as more SNPs are added, or if they plateau or decline, as discussed in Section 2.5. The results guided decisions on feature set size in subsequent experiments.

3.7 Feature Extraction with PCA and Autoencoder

In Section 2.6 we introduced autoencoders. Here, we describe the specific autoencoder architecture employed in our experiments and explain how we used embeddings from both PCA and the autoencoder to make predictions.

The general design of our autoencoder was inspired by Ausmees and Nettelblad (2022) and their so-called Genotype Convolutional Autoencoder (GCAE), developed explicitly for nonlinear dimension reduction of SNP data. However, our final structure (Figure 10) evolved through trial and error, guided by embedding visualizations and preliminary regression results. The final architecture employed 1D convolutional layers with eight filters, each scanning the input to detect local patterns. As a result, the output from the convolutional layers was a set of eight sequences, one for each filter. Then, pooling was applied to each sequence, reducing the lengths by half. Dense layers formed a bottleneck at the center, where the input was compressed into a low-dimensional latent representation. After the latent layer in the middle, the decoder mirrored the encoder with upsampling instead of pooling to restore the original shape. The upsampling layer duplicated positions along the 1D axis via the nearest neighbor approach. Residual blocks are shown in Figure 10 with dotted lines. Appendix B holds a detailed summary of the architecture, including hyperparameters.

Having decided on a working architecture, we trained multiple autoencoders varying the number of latent dimensions ($[10, 20, 30, 40, 50]$) on a subset of 64,000 SNPs. We applied the same dimensionality settings to the principal components (PCs) from PCA, enabling a direct comparison between linear and non-linear embeddings as input. To asses the

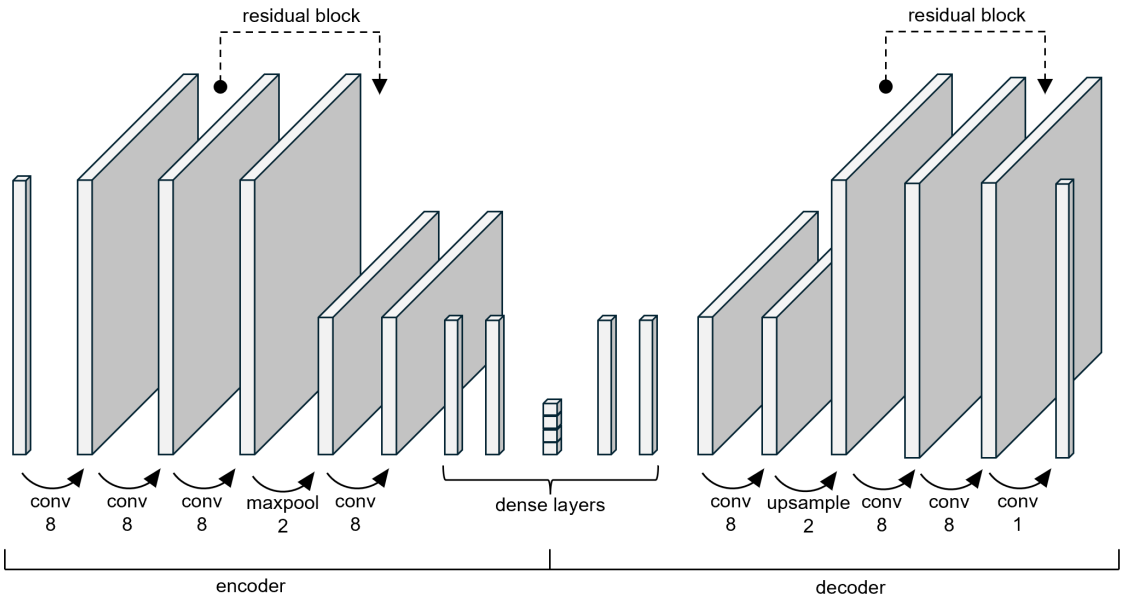


Figure 10: Architecture of the GCAE employed for SNP dimension reduction in this thesis. The encoder (left) compresses the input into a low-dimensional latent space (the bottleneck layer in the middle), while the decoder (right) attempts to reconstruct the input data from the compressed data. The numbers below the layers specify either the filter count for convolutions or the down- or upsampling factor for pooling and upsampling layers. Dotted lines mark residual blocks.

predictive value of these embeddings, we trained CatBoost models using them as input features. CatBoost was chosen for its strong performance without hyperparameter tuning. As a baseline, we also trained CatBoost models directly on the raw SNP subsets to evaluate whether models trained on compressed representations could match or outperform those trained on the raw data.

4 Results

4.1 Preliminary Analysis of Adjusted and Mean Phenotypes

In this section, we present descriptive statistics for both the mean and adjusted phenotypes of the sparrows, providing initial insights into the data before proceeding to more complex analyses. By comparing variances of the mean phenotype and the adjusted one (Table 2), we can quantify how much of the phenotypic variation was contributed by the environmental variables in our two-step procedure. For body mass and wing length, the variance falls by about two-thirds, from 4.647 to 1.434 and 5.278 to 1.626, respectively, whereas for tarsus length it falls only marginally, from 0.642 to 0.592. The variance ratio $\text{Var}(\mathbf{y}^*)/\text{Var}(\bar{\mathbf{y}})$ between the adjusted and mean phenotypes provides an upper bound on the proportion of genetic variation V_G for each trait. Any unmeasured environmental effect or gene-by-environment interaction that remains in the adjusted phenotype would reduce the genetic share further. The correlations between the adjusted and mean phenotypes for mass and tarsus (Table 3) remain above 0.95, indicating that the adjustment subtracts an almost uniform shift or rescaling across individuals, with minor reordering. The wing correlation drops to 0.74, suggesting that one or more environmental effects substantially reorder individuals after adjustment. This is informative because it tells us how well the raw phenotypes are represented by the adjusted ones. High correlation implies that a model capable of closely approximating the adjusted phenotype – our actual modelling target – should also predict the raw phenotypes well.

Table 2: Variation in mean phenotypes and adjusted phenotypes for body mass, tarsus length, and wing length.

	Mass Variance	**Tarsus Variance**	**Wing Variance**
Mean phenotype, $\bar{\mathbf{y}}$	4.647	0.642	5.278
Adjusted phenotype, $\mathbf{y}^*$	1.434	0.592	1.626

Table 3: Pearson correlations, $\text{Corr}(\mathbf{y}^*, \bar{\mathbf{y}})$, between adjusted phenotypes $\mathbf{y}^*$ and the mean phenotype $\bar{\mathbf{y}}$ in body mass, tarsus length, and wing length.

Mass Corr	**Tarsus Corr**	**Wing Corr**
0.951	0.988	0.740

4.2 Dimension Reduction and Visualization with t-SNE

We revisit the t-SNE dimension reduction results from Wold (2024) (Figure 11), where data points are color-coded by hatch island. To decrease noise, only the eight islands with the largest sample sizes were included in this analysis. The t-SNE projections reveal a notable degree of clustering associated with the sparrows' hatch islands, indicating a corresponding genetic structure. While individuals from the same islands cluster together, substantial overlap between islands remains. Island 38 (Aldra) appears to be the most isolated in the projection.

When examining the migration map in Figure 12, borrowed from Ranke et al. (2018), we observe varying degrees of migration across the islands, contributing to genetic similarity across the meta-population. Notably, only a small migration arrow connects to Aldra, sug-

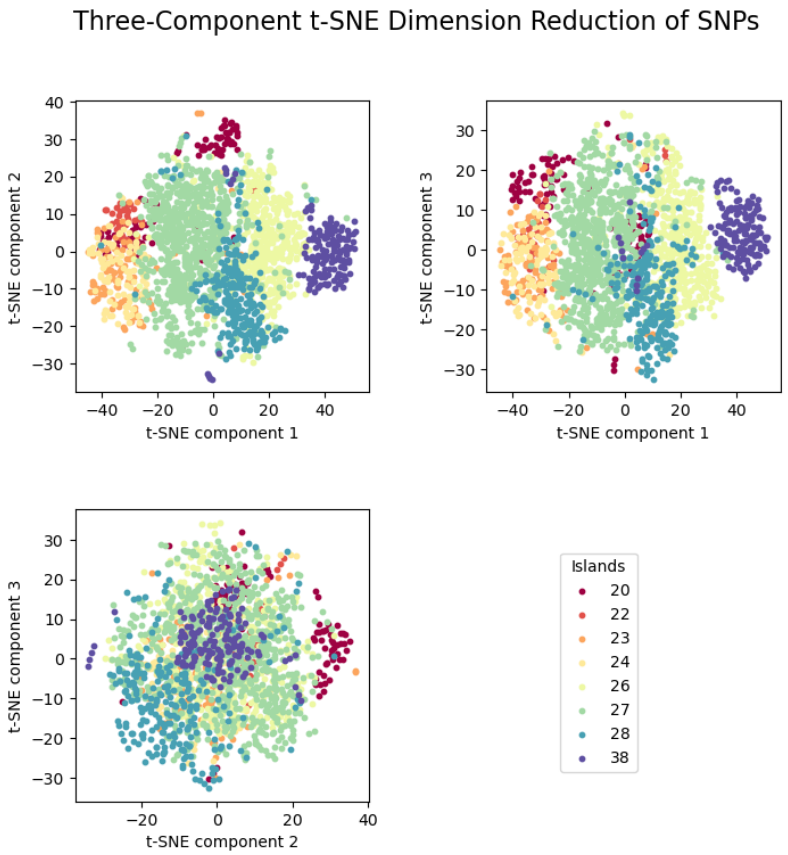


Figure 11: Visualization of a three-component t-SNE dimension reduction of the SNP data, using cosine similarity as the distance metric and perplexity set to 50. The plot highlights clusters based on the sparrows' hatch islands.

gesting limited genetic exchange with other islands; this isolation is reflected in the distinct clustering of Aldra in the t-SNE plot. Conversely, there is significant two-way migration between islands 23 (Træna) and 24 (Selvær), suggesting genetic similarity between these islands. Consistent with this, they appear interchangeable in the t-SNE plot, with island 22 (Myken) also appearing in close proximity. Overall, the clustering patterns correspond well with the migration map.

Assuming genetic similarity is a good predictor for prediction accuracy, one would be prone to believe that a model trained on island 23 will perform better when predicting on island 24 than on island 38, for example. As discussed in Section 2.3.1, population structure, as we have here, can be detrimental to GP models and may inflate variance in prediction accuracies between folds in a cross-validation process. Some splits can be lucky (genetically similar), while others are not. We will keep this in mind when interpreting the following results. Although the two-step approach should in part compensate for population structure, the genetic basis for prediction still varies across subpopulations, and this heterogeneity may affect how well models trained on pooled data generalize.

Name	Number
Nesøy	20
Myken	22
Træna	23
Selvær	24
Gjerøy	26
Hestmannøy	27
Indre Kvarøy	28
Aldra	38

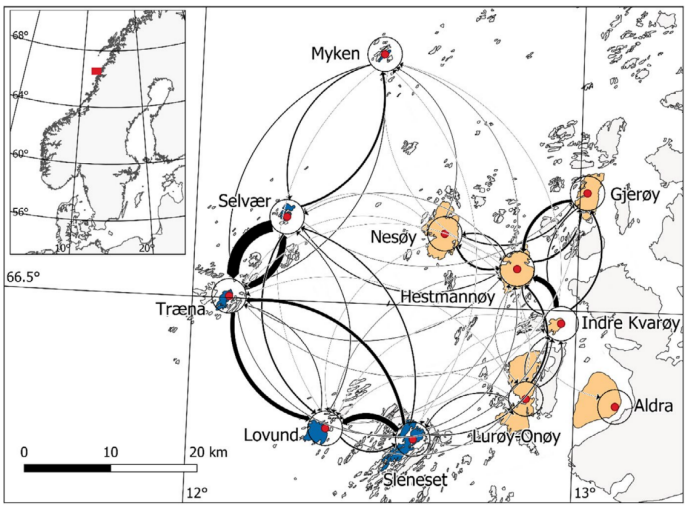


Figure 12: Left: Mapping between names and numbers for the islands used in the t-SNE analysis. Right: Map of the study area. The coloration is the same as in Figure 9. The arrows display emigration patterns in the period 1993-2014. The arrow thickness indicates the emigration volume.

4.3 Baseline Performance with Full SNP Data

We begin by establishing baseline accuracies using the full SNP set (Figure 13), which serves as a reference point for later experiments. The goal is to assess what can be achieved using modern machine learning models and RR without any SNP filtering, and how their performance compares to the animal model (GEBVs). Due to its ability to perform well without tuning, only CatBoost among the gradient boosting models was used at this stage. Given the high computational cost of evaluating gradient boosted models on the complete SNP set, running extensive tuning across multiple algorithms would not be feasible.

The relative performance of the three models varies across phenotypes. Both RR and CatBoost show slightly higher median scores for body mass than the animal model. Still,

the differences are minor, and the animal model's variation across folds broadly overlaps with the other two. This makes it difficult to conclude that either model outperforms the animal model. For tarsus length, the trend flips – CatBoost performs worse and shows greater variability, while RR and the animal model perform similarly. Regarding wing length, the animal model outperforms both alternatives, with RR trailing behind and CatBoost lagging further.

The differences in predictive accuracy across phenotypes are no surprise. One possible explanation is variation in trait heritability – the proportion of variance explained by genetic factors – along with differences in exposure to environmental noise. For instance, body mass may be more influenced by environmental factors not controlled for, making it harder to predict from genetic data alone. Overall, these results provide an initial sense of where gradient boosted methods stand in this setting. RR appears to be competitive with the animal model in two out of three scenarios, suggesting it could be a viable alternative under certain conditions.

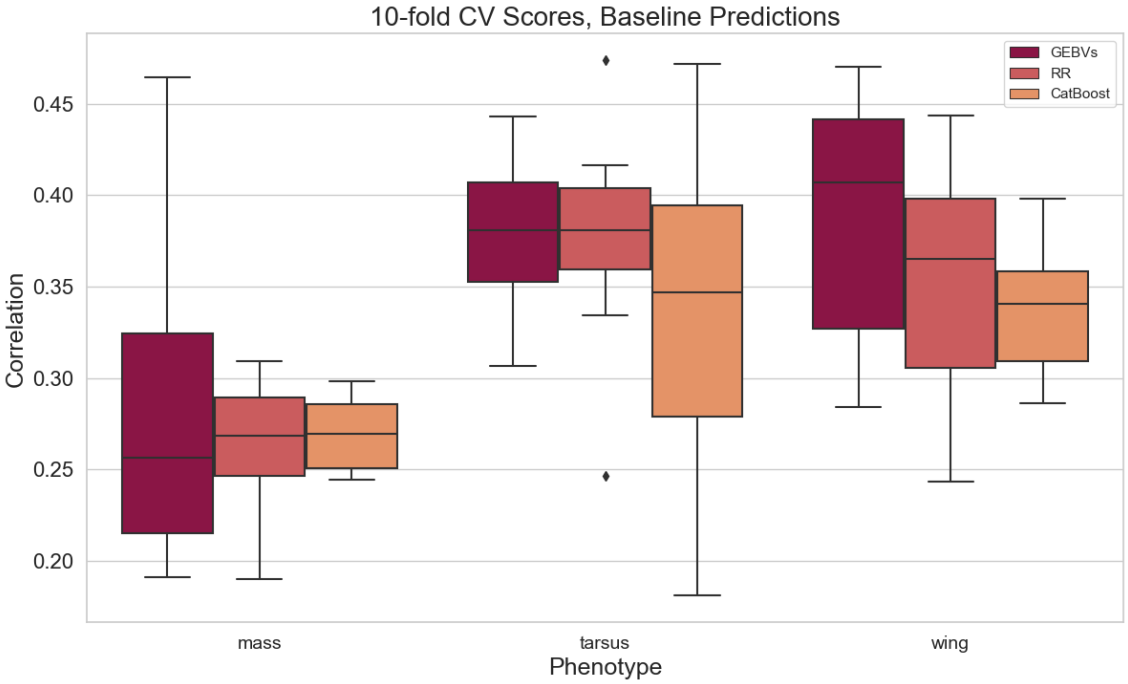


Figure 13: Boxplots of correlation between predicted genetic effects and the mean phenotypes for mass, tarsus, and wing with all SNPs used. The GEBVs are predictions from the animal model, brought on from Wold (2024). RR and CatBoost are predictions made using the respective models with all SNPs. X-axis: Phenotype. Y-axis: Pearson correlation between the predicted values and the mean phenotypes.

4.4 Selecting Subsets of SNPs

Figure 14 presents the results of the subset selection experiment, illustrating how model accuracy changes with increasing numbers of SNPs. To our knowledge, no previous studies have presented such a comprehensive analysis of predictive performance across this wide a range of SNP subset sizes on this dataset, making these results particularly interesting. Our primary interest is understanding how accuracy evolves as we increase the subset sizes. First, we observe that accuracy levels differ among the three phenotypes, within both

model frameworks. Consistent with the baseline predictions (Figure 13), the accuracies for tarsus and wing are generally higher than for mass. RR shows a clear and consistent trend; accuracy improves steadily for all traits as more SNPs are added. This aligns with expectations – bias decreases with more features, while RR's regularization keeps variance down. However, the magnitude of improvement differs between traits: while RR shows large gains for tarsus and wing as SNP counts increase (approximately 0.07 and 0.06 from 10,000 to 180,000, respectively), the improvement for mass is more modest (around 0.03).

In contrast, the gradient boosted trees (CatBoost) display less predictable patterns. For mass, however, CatBoost exhibits a notable rise in accuracy, peaking at around 60,000 SNPs, followed by a decline and increased variability beyond that point. This resembles a textbook example of the bias-variance tradeoff discussed in Section 2.5, where adding more features eventually leads to overfitting and performance degradation. Interestingly, this pattern is not observed for the other two traits. For tarsus, while the highest median accuracy appears at 80,000 SNPs, there is no clear upward trend before the peak, nor a consistent decline after. The curve appears more erratic than for mass. Wing shows a similarly inconsistent pattern, with relatively high median values at both 70,000 and 160,000 SNPs. Notably, CatBoost outperforms RR only for the mass phenotype, reinforcing the baseline result from Section 4.3. This raises interesting questions about trait architecture. It may be that mass, more than the other two, is driven by non-linear effects that regression trees can capture, whereas for tarsus and wing, linear models may be more appropriate. We will return to this in the discussion (Section 5).

A key takeaway from this experiment is that – despite the different patterns observed – none of the phenotypes show substantial gains in accuracy beyond 60,000 to 80,000 SNPs. While it is unclear whether that many markers are strictly necessary for tarsus and wing, it appears that around 60,000 SNPs is the best number to achieve peak performance with CatBoost for mass. As discussed in Section 2.5, one of our goals was to determine whether a reduced SNP set could yield equal or better predictions than the complete set of 182,854 markers. These results suggest that, at least for tree-based models, using too many SNPs can degrade accuracy, particularly for mass. Based on our findings, all subsequent experiments are conducted using a maximum of 64,000 SNPs. This should result in good predictive performances and will significantly reduce the computational cost of training models when we include the other gradient boosting methods combined with hyperparameter optimization.

4.5 Tuned Gradient Boosted Models on Reduced SNP Set

Using 64,000 SNPs, we trained all gradient boosted models with hyperparameter optimization to obtain the best possible predictive performance (Figure 15). Consistent with earlier results, the gradient boosted models are competitive with both the animal model and RR for the mass phenotype but seem to lag slightly behind for tarsus and wing. Notably, all models for mass and tarsus (excluding RR, which was not included in the earlier study) show a substantial improvement compared to those trained on the smaller 15,000 SNP subset used in Wold (2024). In contrast, improvements for wing are only marginal. These gains suggest that increasing the SNP set to 64,000 significantly affected predictive power – at least for mass and tarsus. Across the different gradient boosting frameworks, prediction accuracies are broadly similar, with no single method clearly outperforming the others. While the gradient boosted models appear competitive – particularly for mass – there is still considerable overlap in their accuracy distributions, with no clear performance differences between them. The animal model, however, appears at least as strong,

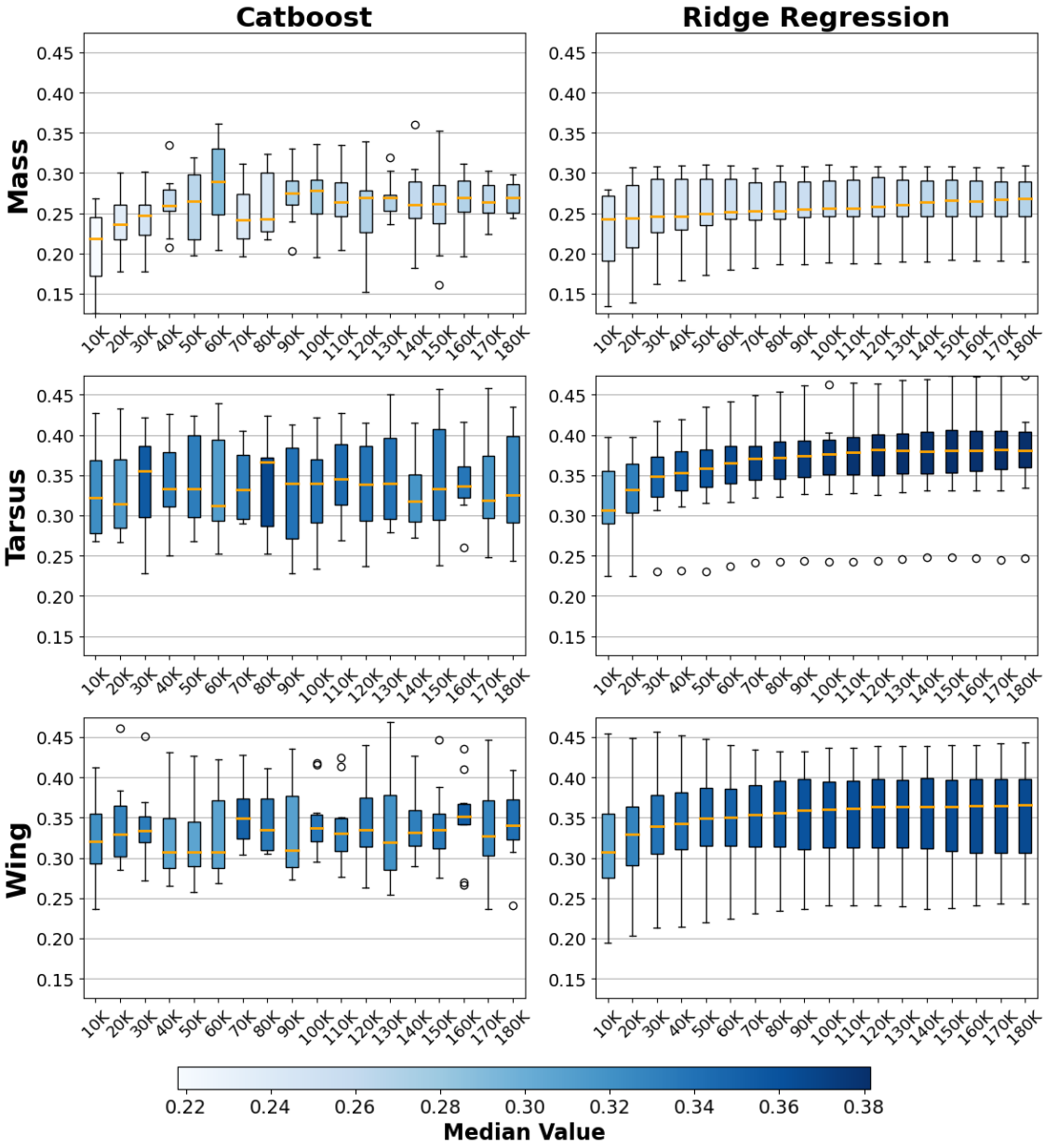


Figure 14: Box-plots illustrating the dependency of the accuracy on the number of sampled SNPs, distributed over phenotype and type of model used. X-axis: Number of SNPs used in the subset. Y-axis: Pearson correlation between the predicted values and the mean phenotypes for each individual. The orange line marks the median value for each distribution.

and potentially more robust, across phenotypes.

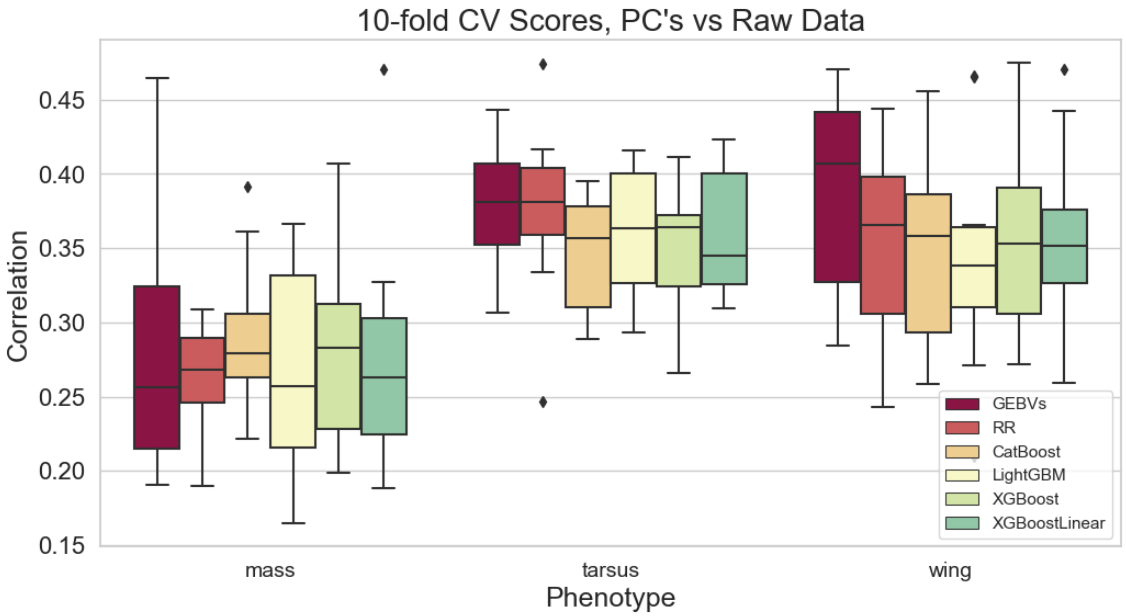


Figure 15: Boxplots of correlation between predicted genetic effects and the mean phenotypes for mass, tarsus, and wing. The GEBVs and RR are the same as in Figure 13, while CatBoost, LightGBM, XGBoost, and XGBoost-Linear are gradient boosted models fitted and hyperparameter optimized with 64,000 SNPs. XGBoostLinear uses linear base learners.

4.6 Regression with Compressed Data

Fitting the Autoencoder

The first step in this experiment was to design the architecture of the autoencoder. During development, we initially trained the network to produce three-dimensional embeddings to allow for visual inspection of the latent space. This provided a fast and intuitive way to assess whether the model had learned meaningful structure: poor fits typically resembled random scatter, while well-trained models produced clear clustering that reflected known population structure, as in Figure 16. Clear patterns emerge when color-coding each individual by hatch-island, with individuals clustering according to their genetic background. Notably, individuals from island 38 (Aldra) form a distinct purple cluster, while islands 23 (Træna) and 24 (Selvær) are closely intertwined, with island 22 (Myken) nearby. Individuals from islands 20, 26, 27, and 28 also tend to group, forming relatively distinct clusters. These results are consistent with the t-SNE visualizations in Section 4.2, suggesting that the autoencoder is learning a meaningful representation of the data in three dimensions. The next step was to increase the latent dimensions and evaluate predictive performance using these embeddings as input to a tree-based boosted model, alongside PCA-based features.

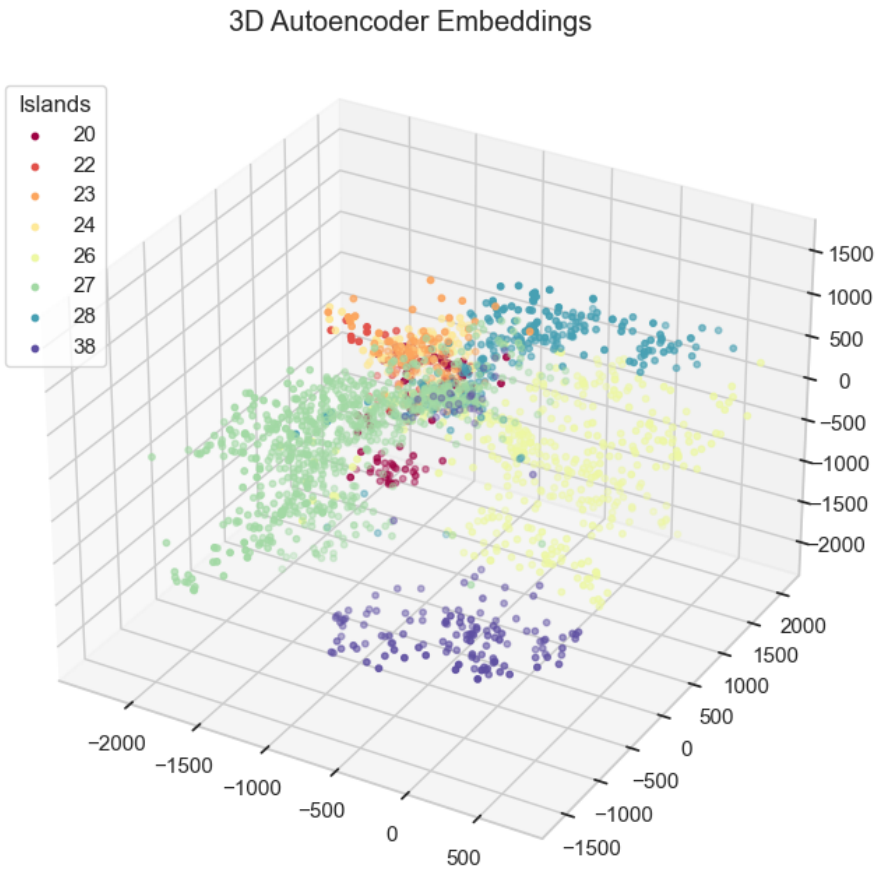


Figure 16: Visual representation of a three-dimensional latent field made by compression of 16K SNPs with the autoencoder. As in Figure 11, only the eight islands with the largest sample sizes are represented in the plot.

PCA vs Autoencoder

The results from CatBoost models trained on embeddings from both the autoencoder and PCA, alongside models trained on raw SNPs, are shown in Figure 17. When restricted to only 10 components, the non-linear embeddings appear to be more informative than the PCs, particularly for tarsus and wing. However, this advantage disappears quickly, accuracy does not improve as more components are added. In contrast, models trained on PCs show a more consistent improvement in accuracy as the number of components increases (for tarsus and wing), while performance for mass remains relatively flat. Overall, the PCs appear to provide more informative features for prediction, despite the autoencoder's theoretical advantage in capturing non-linear structure. PCA is deterministic and robust and requires no tuning, consistently extracting components that explain the most variance in the input data. Autoencoders, by contrast, are sensitive to architecture and hyperparameter settings, and our implementation was likely not optimally configured for this dataset.

Based on median accuracy, models trained on raw SNPs outperform those trained on compressed features for all phenotypes. This finding was expected, given that we limited the embeddings to relatively few dimensions. Dimension reduction inevitably discards some genetic variation relevant for prediction. Prior studies often use hundreds of principal components (Macciotta et al., 2010; Hosseini-Vardanjani et al., 2018), far more than used here. In this initial experiment, we aimed to explore whether non-linear embeddings from an autoencoder could offer stronger features for GP with tree-based models. That potential advantage did not materialize here, suggesting that either the linear PCs are better or that the autoencoder used was not optimal for this task.

Ridge Regression and CatBoost with All Principal Components

Suspecting that further performance could be gained by using more PCs than shown in Figure 17, we trained RR and CatBoost models on the full set of PCs, equal to the number of individuals. We did not do the same for the autoencoder, as those would be much more expensive to compute and unlikely to outperform PCA based on previous results. Figure 18 compares RR and CatBoost trained on the full set of PCs (derived from all 182,854 SNPs) to models trained on the full raw SNP matrix. RR performs nearly as well with PCs as with raw SNPs, while CatBoost shows a noticeable drop in accuracy, performing even worse than with only 10 PCs. Potential explanations are discussed in Section 5.

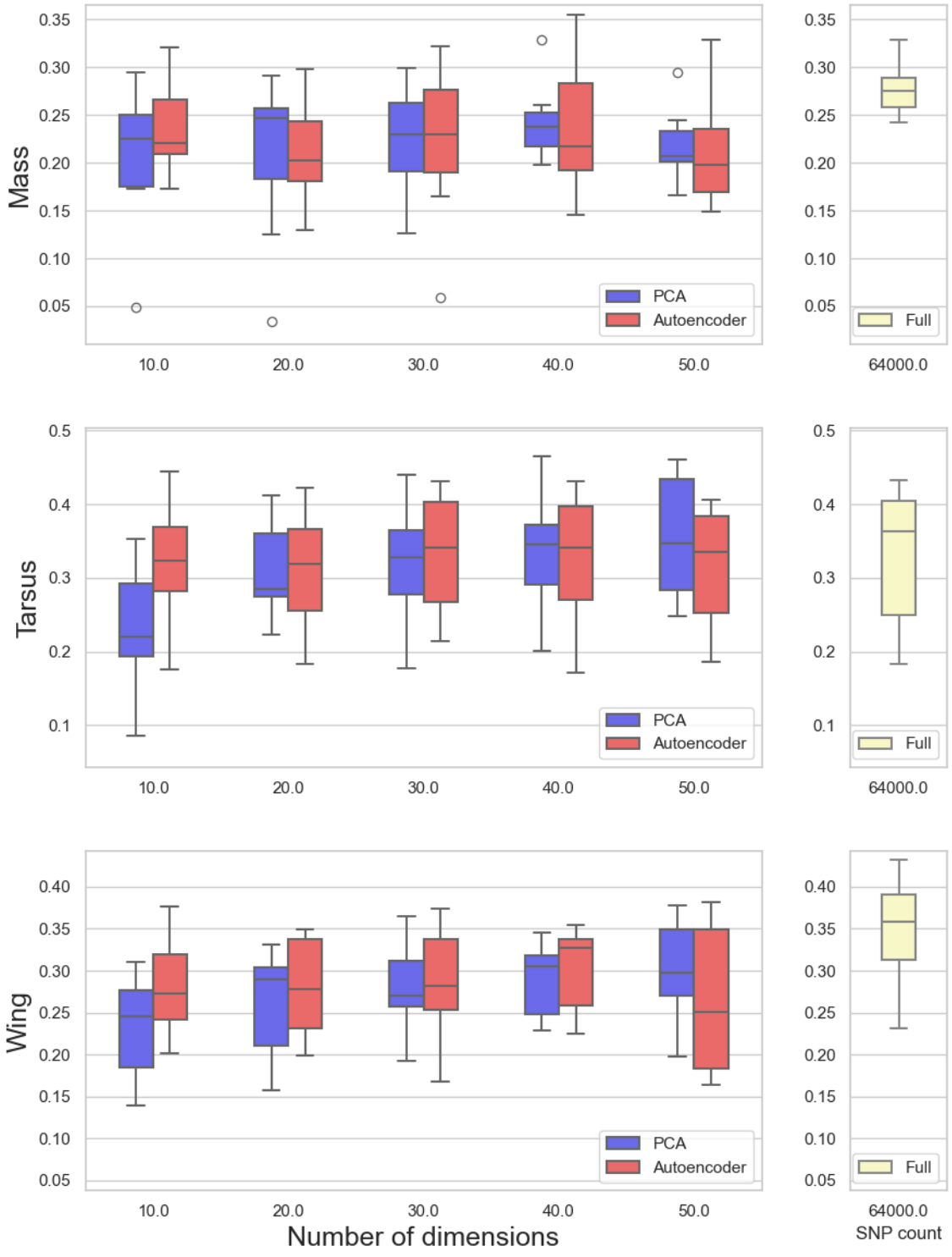


Figure 17: Comparison of accuracy using compressed SNPs. Left: CV-scores when modeled with compressions from PCA and autoencoder. The number of dimensions (x-axis) varied from 10 to 50, using 64,000 SNPs. Right: For reference, accuracies using the full, uncompressed 64,000 SNPs.

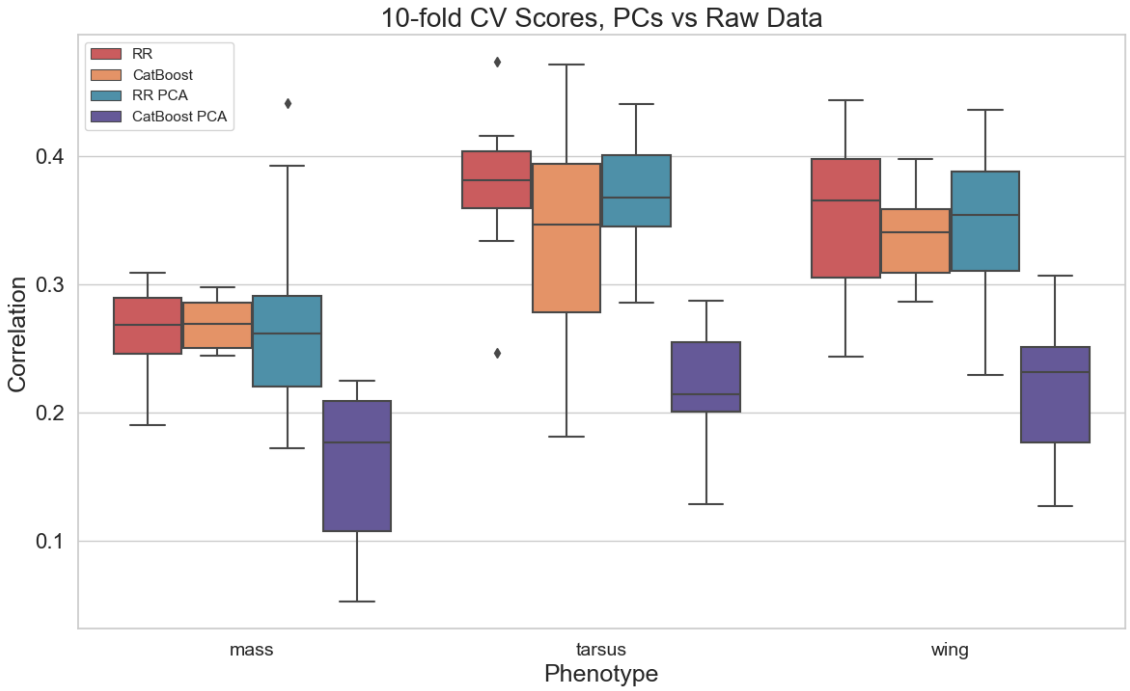


Figure 18: CV-scores for the three phenotypes using four modeling strategies: RR, CatBoost, RR with PCs, and CatBoost with PCs (the last two named RR PCA and CatBoost PCA in the legend). All SNPs are used for all models. Phenotypes along the x-axis and predictive accuracy along the y-axis.

5 Discussion

In this thesis, we have explored the performance of advanced ML techniques alongside RR for GP, specifically aiming to determine whether these approaches can match or exceed the performance of the established Bayesian animal model. All analyses were conducted on data from a wild house sparrow population inhabiting an island system along the coast of northern Norway. Building on the work of Wold (2024), we focused on three quantitative traits – body mass, tarsus length, and wing length – using both subsets of raw SNPs and features derived from dimension reduction techniques. Our approach employed a two-step modeling procedure: first, constructing an adjusted phenotype from which measured environmental effects were removed, then modeling this adjusted phenotype using RR and gradient boosted models (XGBoost, LightGBM, and CatBoost). While LightGBM and CatBoost utilized regression trees, XGBoost was evaluated using both regression trees and linear base learners.

Suspecting structural genetic differences between subpopulations in our sparrow data, we employed t-SNE for dimensionality reduction and visualization. The t-SNE plot (Figure 11) revealed distinct genetically based clusters that align well with the previously known migration patterns (Figure 12). Islands with substantial gene flow appear genetically intertwined, whereas isolated islands form distinct genetic clusters. Such a structure can induce high variance in accuracy over a cross-validation process, as subpopulations are randomly distributed over folds, meaning they could be highly over- or underrepresented in some folds. This may partially explain the substantial variability in prediction accuracies observed across folds in our models.

With the aim of reducing computational effort and investigating a potential bias-variance tradeoff, we conducted a subset selection experiment using randomly sampled SNP subsets with CatBoost and RR, ranging from 10,000 up to the full set of 180,000 SNPs. A notable finding was that CatBoost accuracy for body mass clearly peaked around 60,000 SNPs, suggesting an optimal subset size due to the bias-variance tradeoff. No similar peak was observed for tarsus and wing. Regardless, these findings led us to select a subset of 64,000 SNPs for subsequent analyses with the gradient boosted models, significantly higher than the 15,000 SNP subset employed in earlier work. In contrast to CatBoost, RR showed a steady accuracy increase as more SNPs were added across all traits, though the gains were smaller for body mass. This steady improvement may be explained by the regularization of RR, which shrinks coefficients and limits overfitting as the penalty term restricts the influence of weak or noisy SNPs.

Utilizing the selected 64,000 SNPs, we conducted hyperparameter optimization for all gradient boosted models, resulting in substantial accuracy improvements for two phenotypes. These results suggest that around 60,000 SNPs may effectively balance capturing genetic signal and avoiding overfitting, reflecting a favorable tradeoff between model complexity and predictive accuracy for gradient boosted trees. Nevertheless, the Bayesian animal model remained the strongest overall performer, producing consistently high accuracies across all traits. A potential advantage of the animal model is its use of known relatedness via the GRM. In contrast, our two-step approach discards the explicit correlation structure and assumes independence among individuals in the adjusted phenotype – an assumption we know to be false. That said, other studies, such as Ashraf et al. (2021), have shown that BayesR, which also uses the two-step approach, tends to outperform the animal model.

Another key factor is how well the assumptions of the models align with the underlying

genetic architectures of the traits. Highly polygenic traits – influenced by many small effects spread across the genome – are particularly well suited to models like the animal model, which aggregate weak signals using the GRM. Ridge regression also benefits from polygenic traits, applying uniform shrinkage and assuming all features contribute modestly. When the signal is so diffuse that no single SNP stands out, the animal model – instead of looking for meaningful SNPs – relies on the assumption that genetically similar individuals will have similar phenotypes. However, if only a few SNPs have big effects, knowing the overall relatedness of individuals might not be very informative. In that case, models that can find and exploit those few SNPs directly, like tree-based models, can be better. These differences may help explain the variation in relative model performance between traits.

In addition to the number and size of contributing SNP effects, the form of those effects – linear or non-linear – is another influential factor. It is known that the trees do not capture the additive effects as well as the linear models, meaning that if a trait is predominantly additive, the boosted trees perform worse than the linear models. Decision trees approximate relationships through a series of discrete splits, producing piecewise constant functions that can only mimic linear trends in a rough, stepwise fashion (Hastie et al., 2009). Even when aggregated in ensembles like gradient boosting, this approach remains less efficient than directly modeling linear effects with a linear model. The true strength of the trees lies in capturing non-linear patterns and interactions. The implication is that the GEBVs and the tree-based models might capture somewhat different aspects of the genetic variance. However, by this explanation, XGBoost with the linear base learner should be closer to RR and the GEBVs than the rest for tarsus and wing, which it is not. The key takeaway may be that different traits might benefit from different models, depending on their genetic architecture. Overall, our results demonstrate that the animal model remains the superior model, consistently matching or exceeding RR and the gradient boosted models.

Throughout all results, prediction accuracies were consistently higher for tarsus and wing length than for body mass, suggesting differences in trait heritability. Compared to relatively stable morphological traits like tarsus and wing length, body mass is a more dynamic trait, responding rapidly to short-term environmental and physiological factors. Such short-term fluctuations – caused by feeding, drinking, or defecating – may not be biologically meaningful in our setting, but they still contribute to the total phenotypic variance (Ponzi et al., 2018). The added variance reduced the proportion of variance directly attributable to genetic factors, making body mass a more challenging target for genomic prediction, regardless of the modeling approach.

To extend the subset selection analysis, we explored dimensionality reduction techniques to compress 64,000 SNPs into a smaller set of features. Using PCA and a non-linear autoencoder, the goal was to create compact representations that could reduce noise and prevent overfitting when used as input to CatBoost. While the autoencoder has the potential to capture non-linear structure, its embeddings did not outperform PCs when used as input to CatBoost models overall. However, when limited to just 10 features, the autoencoder embeddings appear to be better input than the corresponding PCs. Several factors may explain these results. First, deep learning models generally require large sample sizes to train effectively. Given our relatively small dataset, the autoencoder may have struggled to generalize. Further, the autoencoder's architecture was initially designed for 3D visual representations; although it performed meaningfully in that context, it may not generalize well to higher-dimensional latent spaces without architectural adjustments. Unlike PCA, which deterministically captures directions of maximum variance, autoencoders are sensitive to hyperparameters and network design. Tuning them effectively for genomic data

is a non-trivial task and may deserve its own study. Moreover, while adding more PCs tends to improve performance – since each additional component introduces new variance – autoencoders do not guarantee such incremental benefits. Because the network is trained to compress all relevant variation into the limited latent space available, simply increasing the number of latent dimensions may not yield more informative features unless the architecture is appropriately scaled. Judging by the accuracies in Figure 17, it does not look like the autoencoder captured meaningful additional information going from 10 to 50 components. In contrast, the PCA-based models steadily improved as more components were added, demonstrating that relevant genetic variation is spread across many components, particularly for tarsus- and wing length. Interestingly, no systematic improvement was observed for mass, where performance remained relatively flat beyond the first 10 PCs.

While the embeddings from the autoencoder did not prove any better than PCs when used as input, the steady improvement in accuracy with increasing numbers of PCs suggested that further predictive gains might still be achievable by including additional components. We therefore conducted a follow-up experiment in which all available PCs derived from the full SNP matrix were used as input features for RR and CatBoost models. RR showed only a modest drop in accuracy when trained on all PCs compared to using raw SNPs. Although PCA retains the total variance of the dataset, it still projects the data into a lower-dimensional space – discarding over 180,000 original SNP features. Some information loss is thus expected. However, RR is designed to handle high-dimensional data through its own regularization and does not require the dimensionality reduction that PCA provides. The slight performance difference may reflect the preservation of weak or distributed effects in the raw data not fully captured in the PCs. CatBoost, by contrast, performed considerably worse when trained on all PCs – worse even than when using only the first 10 components – highlighting a limitation of using PCs as input variables in tree-based models. Trees excel at capturing interaction terms, but since PCs mix the effects of many SNPs, it is unclear what the interaction between two PCs means. When using SNPs, interactions are biologically grounded, and any SNP can interact directly with another SNP. As a result, using PCs removes one of the biggest advantages of tree-based models by limiting their flexibility.

In this thesis, we aimed to determine if advanced ML techniques can match or outperform the well-established animal model in genomic prediction. We anticipated that modern approaches, such as gradient boosted trees, might capture both linear and non-linear genetic effects effectively. Additionally, we included RR as a simpler and computationally lighter alternative, evaluating whether the complexity of gradient boosting or Bayesian animal models is justified by improved prediction accuracy over a simpler model. Our analyses revealed that no single model consistently emerged superior across all scenarios. Instead, the relative performance varied among phenotypes, emphasizing the importance of trait-specific genetic architecture. We cannot rule out the possibility that results could differ significantly in either direction with other phenotypes. Regardless, this variability suggests practitioners may benefit from tailoring their model selection based on genetic properties inherent to each trait, rather than relying on a universal approach. Furthermore, our findings indicate that gradient boosted trees may exhibit an optimal number of SNPs for specific traits, balancing the bias-variance tradeoff. Consequently, SNP subset selection strategies might enhance the performance of these models. In exploring feature extraction methods, PCs did not provide significant predictive benefits over raw SNPs for either RR or the tree-based model. Although neural-network-based non-linear compression via autoencoders produced visually meaningful embeddings in three-dimensional space, achieving effective compression likely demands datasets larger than those typically feasible in wild populations.

Despite the insights gained in this thesis, several questions remain unanswered regarding the underlying causes of our findings and how the current models might be improved. With more time, a natural next step would be to explore more sophisticated SNP selection strategies. For instance, working with the same data as us, Gravdal (2025) showed that training an XGBoost model using only the 10,000 SNPs most correlated with the phenotype yielded performance comparable to using the full SNP set. This finding suggests that more intelligent feature selection could improve accuracy while reducing model complexity. Another exciting possibility would be to attempt to bypass or improve the current two-step approach, which discards the known genomic relationships between individuals across generations. Ideally, the GRM would be embedded directly into the ML models. Recent studies have begun to do this using Graph Neural Networks (GNNs), which effectively incorporate relatedness information from the GRM in genomic prediction tasks, displaying promising results (Kihlman et al., 2024; Zhao et al., 2025). To our knowledge, this has not yet been applied in wild populations, making it a new and potentially impactful path for future research.

In conclusion, our results demonstrate that model performance in genomic prediction is shaped by both population structure, the number of SNPs employed, and the underlying genetic architecture of the traits. Gradient boosted models showed competitive performance against the animal model for specific phenotypes. However, the animal model generally remained the most consistently accurate method. Notably, RR emerged as a robust, efficient, and computationally less demanding alternative, performing comparably to or better than gradient boosted models across all traits. Given its ease of implementation and computational efficiency, RR may represent an advantageous choice for genomic prediction tasks in wild populations compared to the gradient boosted models.

Bibliography

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework.

Ashraf, B., Hunter, D. C., Bérénos, C., Ellis, P. A., Johnston, S. E., Pilkington, J. G., Pemberton, J. M., and Slate, J. (2021). Genomic prediction in the wild: A case study in Soay sheep. *Molecular Ecology*, 31(24):6541–6555.

Ausmees, K. and Nettelblad, C. (2022). A deep learning framework for characterization of genotype data. *G3: Genes, Genomes, Genetics*, 12(3).

Barreto, C. A. V., das Graças Dias, K. O., de Sousa, I. C., Azevedo, C. F., Nascimento, A. C. C., Guimarães, L. J. M., Guimarães, C. T., Pastina, M. M., and Nascimento, M. (2024). Genomic prediction in multi-environment trials in maize using statistical and machine learning methods. *Nature*, 14(1).

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bérénos, C., Ellis, P. A., Pilkington, J. G., and Pemberton, J. M. (2014). Estimating quantitative genetic parameters in wild populations: A comparison of pedigree and genomic approaches. *Molecular Ecology*, 23(14):3434–3451.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. Technical report.

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bourlard, H. and Kamp, Y. (1988). Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics*, 59:291–294.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Technical report.

Conner, J. K. and Hartl, D. L. (2004). *A Primer of Ecological Genetics* . Sinauer Associates, Sunderland, 1 edition.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function*. *Math. Control Signals Systems*, 2:303–314.

Darwin, C. (1859). Variation Under Nature. In *On the Origin of Species*, chapter 2. 1 edition.

De Los Campos, G., Gianola, D., and Allison, D. B. (2010). Predicting genetic predisposition in humans: The promise of whole-genome markers. *Nature Reviews Genetics*, 11(12):880–886.

De Vlaming, R. and Groenen, P. J. (2015). The current and future use of ridge regression for prediction in quantitative genetics. *BioMed Research International*, 2015.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Falconer, D. S. and Mackay, T. F. (1989). *Introduction to Quantitative Genetics*. Longman Group, 3 edition.

Fernandes de Mello, R. and Antonelli Ponti, M. (2018). *Machine Learning: A Practical Approach on the Statistical Learning Theory*. Springer.

Frazier, P. I. (2018). A Tutorial on Bayesian Optimization. Technical report.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Technical report.

Garnett, R. (2023). *Bayesian Optimization*. Cambridge University Press, Cambridge, 1 edition.

Gill, M., Anderson, R., Hu, H., Bennamoun, M., Petereit, J., Valliyodan, B., Nguyen, H. T., Batley, J., Bayer, P. E., and Edwards, D. (2022). Machine learning models outperform deep learning models, provide interpretation and facilitate feature selection for soybean trait prediction. *BMC Plant Biology*, 22(1).

Goddard, M. E., Kemper, K. E., MacLeod, I. M., Chamberlain, A. J., and Hayes, B. J. (2016). Genetics of complex traits: Prediction of phenotype, identification of causal polymorphisms and genetic architecture. *Proceedings of the Royal Society B: Biological Sciences*, 283(1835).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gravdal, G. W. (2025). Machine Learning Approaches to Genomic Prediction and Marker Association. Technical report, Norwegian University of Science and Technology.

Griffiths, A. J., Lewontin, R. C., Gelbart, W. M., Suzuki, D. T., Wessler, S. R., and Miller, J. H. (2004). *An Introduction to Genetic Analysis*. Freeman, W.H, 8 edition.

Habier, D., Fernando, R. L., and Dekkers, J. C. (2007). The impact of genetic relationship information on genome-assisted breeding values. *Genetics*, 177(4):2389–2397.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *Elements of Statistical Learning*. Springer, 2 edition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. Technical report, Microsoft Research.

Hellwege, J. N., Keaton, J. M., Giri, A., Gao, X., Velez Edwards, D. R., and Edwards, T. L. (2017). Population Stratification in Genetic Association Studies. *Current Protocols in Human Genetics*, 95(1):1–1.

Henderson, C. R. (1975). Best Linear Unbiased Estimation and Prediction under a Selection Model. Technical Report 2.

Henderson, C. R. (1984). Applications of Linear Models in Animal Breeding. Technical report.

Hosseini-Vardanjani, S. M., Shariati, M. M., Shahrebabak, H. M., and Tahmoorespur, M. (2018). Incorporating prior knowledge of Principal Components in genomic prediction. *Frontiers in Genetics*, 9.

Hughes, G. F. (1968). On the Mean Accuracy of Statistical Pattern Recognizers. *IEEE Transactions on Information Theory*.

Hunter, D. C., Ashraf, B., Bérénos, C., Ellis, P. A., Johnston, S. E., Wilson, A. J., Pilkington, J. G., Pemberton, J. M., and Slate, J. (2022). Using genomic prediction to detect microevolutionary change of a quantitative trait. *Proceedings of the Royal Society B: Biological Sciences*, 289(1974).

International Society of Genetic Genealogy (2024). Single-nucleotide polymorphism.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Technical report, Google Inc.

James, G., Witten, D., Hastie, T., Tibshirani, R., and Taylor, J. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer, 1 edition.

Jehan, T. and Lakhanpaul, S. (2006). Single nucleptide polymorphism (SNP)-Methods and applications in plant genetics: A review. *Indian Journal of Biotechnology*, 5:435–459.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Technical report.

Kihlman, R., Launonen, I., Sillanpää, M. J., and Waldmann, P. (2024). Sub-sampling graph neural networks for genomic prediction of quantitative phenotypes. *G3: Genes, Genomes, Genetics*.

Kliman, R. M., editor (2016). *Encyclopedia of Evolutionary Biology*. Academic Press, 1 edition.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243.

Kruuk, L. E. (2004). Estimating genetic parameters in natural populations using the 'animal model'. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 359(1446):873–890.

Lasky-Su, J., Lyon, H. N., Emilsson, V., Heid, I. M., Molony, C., Raby, B. A., Lazarus, R., Klanderman, B., Soto-Quiros, M. E., Avila, L., Silverman, E. K., Thorleifsson, G., Thorsteinsdottir, U., Kronenberg, F., Vollmert, C., Illig, T., Fox, C. S., Levy, D., Laird, N., Ding, X., McQueen, M. B., Butler, J., Ardlie, K., Papoutsakis, C., Dedoussis, G., O'Donnell, C. J., Wichmann, H. E., Celedón, J. C., Schadt, E., Hirschhorn, J., Weiss, S. T., Stefansson, K., and Lange, C. (2008). On the Replication of Genetic Associations: Timing Can Be Everything! *American Journal of Human Genetics*, 82(4):849–858.

Li, B., Zhang, N., Wang, Y. G., George, A. W., Reverter, A., and Li, Y. (2018). Genomic prediction of breeding values using a subset of SNPs identified by three machine learning methods. *Frontiers in Genetics*, 9.

Lourenço, V., Ogutu, J., Rodrigues, R. P., Posekany, A., and Piepho, H. P. (2024). Genomic prediction using machine learning: a comparison of the performance of regularized regression, ensemble, instance-based and deep learning methods on synthetic and empirical data. *BMC Genomics*, 25(1).

Lush, J. L. (1943). *Animal Breeding Plans*. Iowa State Collage Press.

Lynch, M. and Walsh, B. (1998). *Genetics and Analysis of Quantitative Traits*. Sinauer Associates, Sunderland, 1 edition.

Macciotta, N. P., Gaspa, G., Steri, R., Nicolazzi, E. L., Dimauro, C., Pieramati, C., and Cappio-Borlino, A. (2010). Using eigenvalues as variance priors in the prediction of genomic breeding values by principal component analysis. *Journal of Dairy Science*, 93(6):2765–2774.

Martino, S. and Riebler, A. (2019). Integrated Nested Laplace Approximations (INLA). Technical report, Norwegian University of Science and Technology.

McGaugh, S. E., Lorenz, A. J., and Flagel, L. E. (2021). The utility of genomic prediction models in evolutionary genetics. *Proceedings of the Royal Society B: Biological Sciences*, 288.

Meuwissen, T., Hayes, B., and Goddard, M. (2016). Genomic selection: A paradigm shift in animal breeding. *Animal Frontiers*, 6(1):6–14.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Mrode, R. A. (2014). *Linear Models for the Prediction of Animal Breeding Values*. CABI Publishing, 3 edition.

National Library of Medicine (2022). What are single nucleotide polymorphisms (SNPs)?

Ponzi, E., Keller, L. F., Bonnet, T., and Muff, S. (2018). Heritability, selection, and the response to selection in the presence of phenotypic measurement error: Effects, cures, and the role of repeated measurements. *Evolution*, 72(10):1992–2004.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2017). CatBoost: unbiased boosting with categorical features. Technical report, Yandex.

Ranke, P. S., Araya-Ajoy, Y. G., Ringsby, T. H., Pärn, H., Rønning, B., Jensen, H., Wright, J., and Sæther, B. E. (2018). Spatial structure and dispersal dynamics in a house sparrow metapopulation. *Journal of Animal Ecology*, 90(12):2767–2781.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing*, volume 1, chapter 8. MIT Press, Cambridge.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, pages 210–229.

Singsaas, H. (2024). Neural networks for genomic prediction in the wild. Technical report, Norwegian University of Science and Technology.

Tian, Y. and Zhang, Y. (2022). A comprehensive survey on regularization strategies in machine learning. *Information Fusion*, 80:146–166.

Trunk, G. V. (1979). Correspondence A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Aanalysis and Machine Intelligence*, 1(3).

United Nations (2015). Transforming our world: the 2030 Agenda for Sustainable Development. Technical report, United Nations.

Van Der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

VanRaden, P. (2008). Efficient Methods to Compute Genomic Predictions. *Journal of Dairy Science*, 91(11).

Visscher, P. M., Hill, W. G., and Wray, N. R. (2008). Heritability in the genomics era - Concepts and misconceptions. *Nature Reviews Genetics*, 9(4):255–266.

Walsh, B. and Lynch, M. (2018). *Evolution and Selection of Quantitative Traits*. Oxford University Press, 1 edition.

Whittaker, J. C., Thompson, R., and Denham, M. C. (2000). Marker-assisted selection using ridge regression. *Genetical Research*, 75(2):249–252.

Wientjes, Y. C., Veerkamp, R. F., Bijma, P., Bovenhuis, H., Schrooten, C., and Calus, M. P. (2015). Empirical and deterministic accuracies of across-population genomic prediction. *Genetics Selection Evolution*, 47(1).

Wold, S. (2024). Genomic Prediction in Wild Populations: Evaluating Gradient Boosting with Reduced SNP Sets. Technical report, Norwegian University of Science and Technology, Trondheim.

Wray, N. R., Kemper, K. E., Hayes, B. J., Goddard, M. E., and Visscher, P. M. (2019). Complex trait prediction from genome data: Contrasting EBV in livestock to PRS in humans. *Genetics*, 211(4):1131–1141.

Zaidi, A. A. and Mathieson, I. (2020). Demographic history mediates the effect of stratification on polygenic scores. *eLife*, 9:1–30.

Zhao, L., Tang, P., Luo, J., Liu, J., Peng, X., Shen, M., Wang, C., Zhao, J., Zhou, D., Fan, Z., Chen, Y., Wang, R., Tang, X., Xu, Z., and Liu, Q. (2025). Genomic prediction with NetGP based on gene network and multi-omics data in plants. *Plant Biotechnology Journal*.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *J. R. Statist. Soc. B*, 67(2):301–320.

Appendix

A Hyperparameter Spaces for Optimization

Appendix A contains tables of hyperparameter spaces used for hyperparameter optimization during model fitting for all models used.

Table 4: The hyperparameters used for tuning XGBoost with regression tree as base learner. The *Distribution* column highlights the distribution each parameter was drawn from per iteration.

Hyperparameter Space, XGBoost		
Name	**Description**	**Distribution**
learning_rate	The learning rate / step size	Random number from a log uniform distribution on $[10^{-3}, 10^{-1}]$
max_depth	Maximum depth of tree	Random integer from $[2, 12]$
subsample	Fracton of observations used for each tree	Random float from $[0.5, 1.0]$
min_child_weight	Minimum sum of instance weight needed in a child	Random integer from $[1, 30]$
lambda	L2 regularization parameter	Random number from a log uniform distribution on $[10^{-2}, 10]$
alpha	L1 regularization parameter	Random number from a log uniform distribution on $[10^{-3}, 1]$
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree	Random number from a log uniform distribution on $[10^{-3}, 1]$

Table 5: The hyperparameters used for tuning LightGBM with regression tree as base learner. The *Distribution* column highlights the distribution each parameter was drawn from per iteration.

Hyperparameter Space, LightGBM		
Name	**Description**	**Distribution**
learning_rate	The learning rate / step size	Random from a log uniform distribution on $[10^{-3}, 10^{-1}]$
max_depth	Maximum depth of tree	Random integers from $[3, 12]$
subsample	Fracton of observations used for each tree	Random float from $[0.5, 1.0]$
min_child_samples	Minimum number of samples needed in a child.	Random integer from $[10, 400]$
reg_lambda	L2 regularization parameter	Random from a log uniform distribution on $[10^{-8}, 10]$
reg_alpha	L1 regularization parameter	Random from a log uniform distribution on $[10^{-8}, 10]$
num_leaves	Maximum number of leaves in one tree	Random integer from $[8, 3000]$

Table 6: The hyperparameters used for tuning CatBoost with regression tree as base learner. The *Distribution* column highlights the distribution each parameter was drawn from per iteration.

Hyperparameter Space, CatBoost		
Name	**Description**	**Distribution**
learning_rate	The learning rate / step size	Random from a log uniform distribution on $[10^{-3}, 10^{-1}]$
depth	Maximum depth of tree	Random integers from $[3, 12]$
subsample	Sample rate for bagging	Random float from $[0.5, 1.0]$
min_data_in_leaf	The minimum number of training samples in a leaf	Random integer from $[10, 400]$
l2_leaf_reg	L2 regularization parameter	Random integer from $[2, 30]$

Table 7: The hyperparameters used for tuning XGBoost with elastic net as base learner. The *Distribution* column highlights the distribution each parameter was drawn from per iteration.

Hyperparameter Space, XGBoostLinear		
Name	**Description**	**Distribution**
learning_rate	The learning rate / step size	Random from a log uniform distribution on $[10^{-3}, 10^{-1}]$
lambda	L2 regularization parameter	Random from a log uniform distribution on $[10^{-4}, 10^{-1}]$
alpha	L1 regularization parameter	Random from a log uniform distribution on $[10^{-4}, 10^{-1}]$

B Architecture of Autoencoder

Appendix B contains tables outlining the architecture of the autoencoder employed in this thesis.

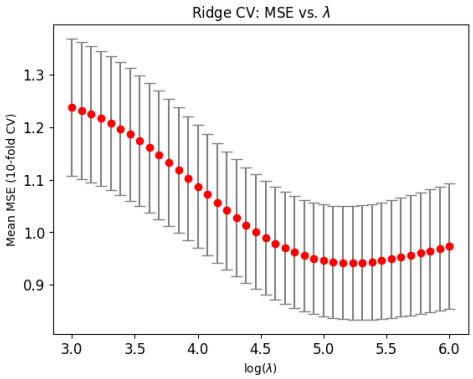
Table 8: Architecture of the employed autoencoder

Encoder	
Layer	**Arguments**
1x Conv1D	filters: 8, kernel size: 5, padding: same, strides: 1, activation: elu
1x BatchNormalization	-
1x ResidualBlock	-
1x MaxPool1D	pool size: 5, strides: 2, padding: same
1x Conv1D	filters: 8, kernel size: 5, padding: same, strides: 1, activation: elu
1x BatchNormalization	-
1x Flatten	-
2x Dropout	rate: 0.1
2x Dense	units: 75, activation: elu
1x Dense (LatentLayer)	units: X

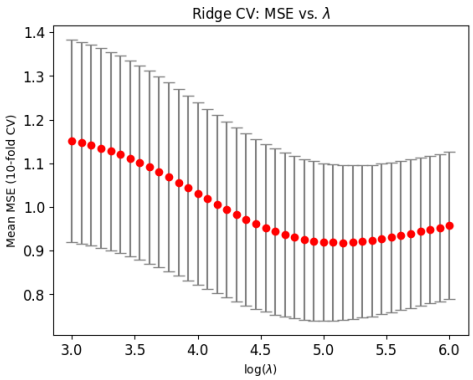
Decoder	
Layer	**Arguments**
2x Dense	units: 75, activation: elu
2x Dropout	rate: 0.1
1x Reshape	target shape: input_length//2, 8
1x Conv1D	filters: 8, kernel size: 5, padding: same, strides: 1, activation: elu
1x BatchNormalization	-
1x UpSampling1D	factor: 2
1x ResidualBlock	-
1x Conv1D	filters: 8, kernel size: 5, padding: same, strides: 1, activation: elu
1x BatchNormalization	-
1x Conv1D	filters: 1, kernel size: 1, padding: same

C Ridge Regression Parameter Search

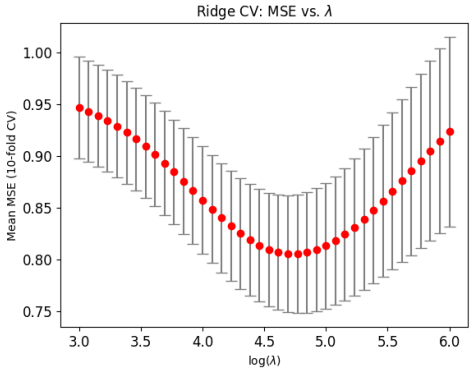
Appendix C contains plots showing the optimization of the regularization parameter (λ) for Ridge Regression across phenotypes. A separate parameter search was performed for each test set in the 10-fold CV. Figure 19 presents the results of one representative fold. Notably, body mass was the only phenotype where the optimal λ-value differed across folds – and even then, only for a single fold.



(a) Mass

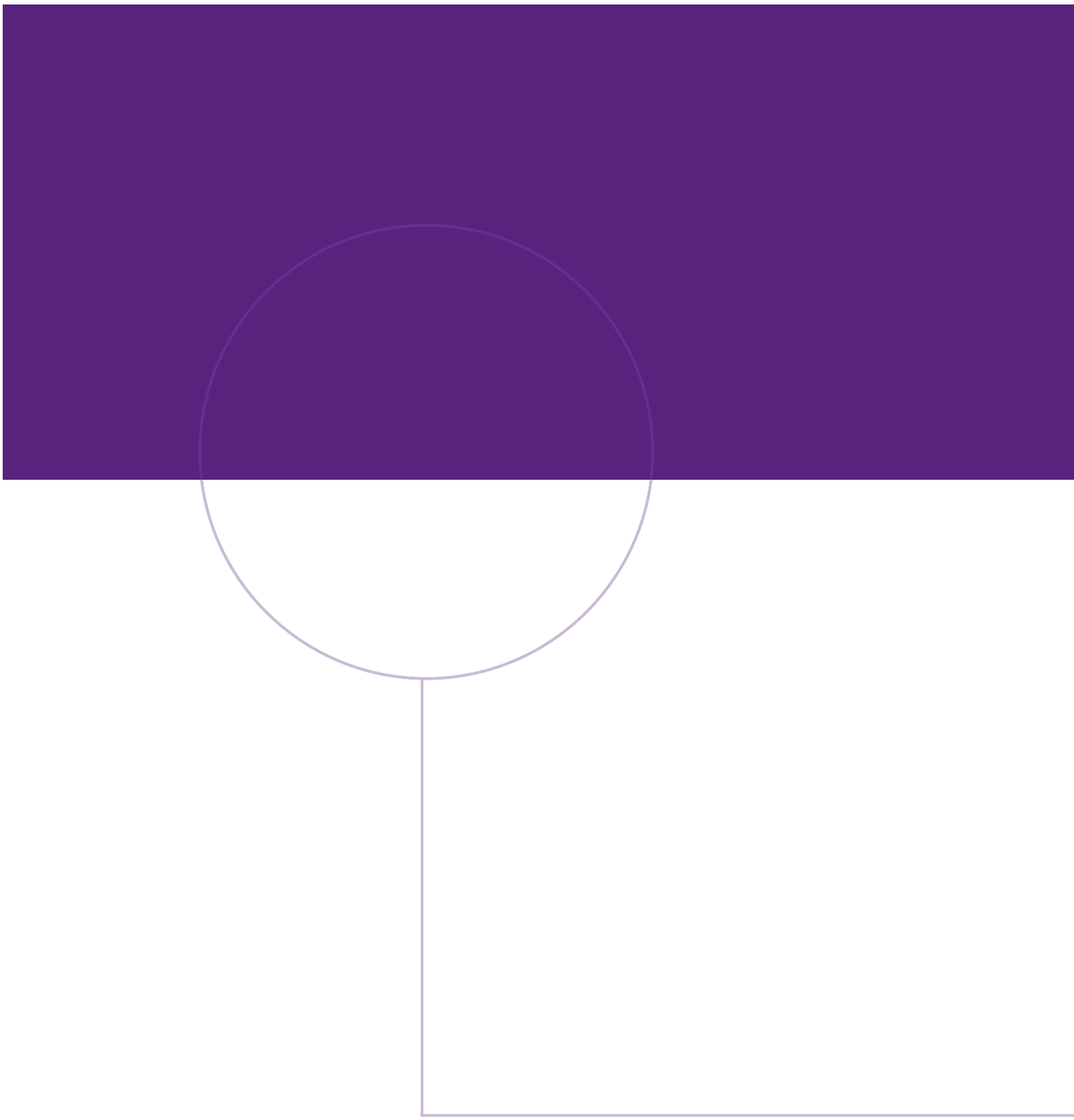


(b) Tarsus



(c) Wing

Figure 19: Parameter search for ridge regression for each phenotype. Models for body mass used $\log(\lambda)$-values of 5.15 and 5.38, while models for tarsus and wing used $\log(\lambda)$-values of 5.15 and 4.74, respectively.



NTNU

Norwegian University of
Science and Technology