NTNU
Kunnskap for en bedre verden

Department of Mathematical Sciences

TMA4500 - Project Thesis

Genotype to Phenotype Predictions Using Boosting Algorithms

Author:
Didrik Lindløv Sand

Autumn 2023

Abstract

Genomic prediction is the study of using statistical approaches to relate an individual's genotypes to its phenotype, and have been dominated by variants of linear regression such as the animal model. In this project we explore a different family of models called boosted regression trees to perform genomic prediction. We test two different strategies to model the phenotype, the first strategy is a two-step model where we first separate the environmental and genetic effects on the phenotype using a mixed model, and then by using boosting algorithms we try to predict the genetic effect on the phenotype. In the second strategy, to allow for interactions between environmental and genetic variables, the environmental and genetic effects are modelled directly together in the boosting algorithms to predict the phenotype. The boosting models are compared to an animal model in both strategies. We investigate two continuous phenotypes, namely bodymass and tarsus length. The data used is from a long term study of house sparrows off the coast of Helgeland in Norway, where genetic data is available in form of SNPs. We use Bayesian optimization to find the best set of hyperparameters for the boosting algorithms, and explore different methods to do feature selection of SNPs prior to training.

We find that boosting algorithms are able to give higher accuracy than the animal model when modelling bodymass for both the two-step strategy and the one-step strategy. With tarsus length as phenotype of interest we find that the boosting algorithms are able to achieve comparable accuracy to the animal model when using the two-step strategy. However, for the one-step procedure with tarsus length as target, the boosting algorithms yield lower accuracies than the animal model. Investigation of tuned hyperparameters values show indications of bodymass being a less polygenetic trait than tarsus length, which may explain the differences in prediction accuracy relative to the animal model.

Table of Contents

List of Figures **iii**

List of Tables **iv**

1 Introduction **1**

2 Background **3**
2.1 Traditional Methods in Ecological Genetics . . . 3
2.1.1 Basic Genetics . . . 3
2.1.2 Quantitative Genetics and Phenotypic Variation . . . 3
2.1.3 The Animal Model . . . 4
2.1.4 BLUP and GBLUP . . . 5
2.2 Bayesian Methods for Animal Models . . . 6
2.2.1 Bayesian Modelling . . . 6
2.2.2 Hierarchical Models . . . 7
2.2.3 Bayesian Animal Model by INLA . . . 8
2.3 Modern Machine Learning Techniques . . . 8
2.3.1 Supervised Learning in General . . . 9
2.3.2 Regression Trees . . . 9
2.3.3 Boosted Regression Trees . . . 11
2.3.4 XGBoost . . . 12
2.3.5 CatBoost . . . 15
2.3.6 LightGBM . . . 17
2.3.7 General Hyperparameters in Boosting Algorithms . . . 18

3 Methods **22**
3.1 Data Description . . . 22
3.2 Genomic Prediction using Boosted-Regression Trees . . . 22
3.2.1 Model Fitting . . . 23
3.2.2 Two-Step Procedure . . . 25
3.2.3 One-Step Procedure . . . 26
3.3 Hyperparameter Tuning . . . 26
3.3.1 General Hyperparamter Tuning . . . 26
3.3.2 Bayesian Optimization . . . 27
3.4 Feature Selection . . . 28

4 Results **32**
4.1 Comparison of Feature Selection Methods . . . 32
4.2 Results From Hyperparameter Tuning . . . 32
4.3 Comparison of Model Performance . . . 35
4.3.1 Two-Step Procedure . . . 35
4.3.2 One-Step Procedure . . . 36

5 Discussion and Conclusions **37**

Bibliography **41**

Appendix **45**
A Hyperparameter Distributions for CatBoost, LightGBM and GBM 45
B Tuned Hyperparameter Values for CatBoost and LightGBM 46

List of Figures

2.1 A simple decision tree 10
2.2 Ordered Boosting 16
2.3 Tree growth strategies 17
2.4 A simple graph 19
2.5 Graph coloring 19

3.1 Conceptual flow of the cross-validation procedure for an arbitrary model . . 25
3.2 Sketch of different hyperparameter strategies. 30

4.1 Comparison of feature selection methods. 33
4.2 Results from two-step procedure 35
4.3 Results from one-step procedure 36

List of Tables

3.1 Hyperparameter Distributions for XGBoost 29

4.1 Tuned hyperparameters for XGBoost 34

1 Hyperparameter Distributions for CatBoost 45

2 Hyperparameter Distributions for LightGBM 46

3 Hyperparameter Distributions for GBM 46

4 Tuned hyperparameters for LightGBM 47

5 Tuned hyperparameters for CatBoost 48

6 Tuned hyperparameters for GBM 48

1 Introduction

The relationship between an individual's genes and its phenotype is a problem that has been subject to research in the fields of evolutionary ecology, animal breeding and crop production a long time (Falconer, 1996; Lynch and Walsh, 1998; Charmantier et al., 2014). For genomic selection, ways to relate breeding values with single nucleotide polymorphisms (SNPs) changed the way breeding is preformed at industry scale (T. H. E. Meuwissen et al., 2001; T. Meuwissen et al., 2016). However, the methods originally developed for agriculture have also been useful in the field of natural populations to assess genetic variation and whether evolution is happening or not (Kruuk et al., 2002; Kruuk, 2004). Evolution happens when allele frequencies change across generations and is often due to natural selection of physical traits (Connor and Hartl, 2004). To understand evolution, and in particular the effect of changes in allele frequencies have on the distribution of the phenotypes, it is important to understand how specific genes relate to specific phenotypes. In addition, this knowledge may also help us understand the relationship between the environment and phenotypes, which will make us more prone to assess consequences of changes in the environment (Chevin et al., 2010).

Traditionally, one has used parametric statistical methods, such as the animal model (Lynch and Walsh, 1998), to model phenotypes as a function of additive gene-expressions and environmental factors, an approach that often neglects the non-additive genetic effects and effects from genetic interactions with the environment, which may have an impact on the phenotype (Munar-Delgado et al., 2023). Additive genetic effects are defined as independent allele contributions. There are two types of non-additive genetic effects that often are the most prominent ones, namely epistatic effects which refer to interactions between genes at different locations (loci) and dominance effects, which are interactions between genes at the same locus (Conner and Hartl, 2004b). Both epistatic and dominance effects refer to interactions between genes, which are referred to as gene-gene interactions. While the environment affects the phenotype directly, such as less food during the winter causing lower body mass, the environment also interacts with the genes. Exposure to certain environments can activate different genes which affects the phenotype, this is known as gene-environment interaction (Virolainen et al., 2022). The "mystery of missing heritability" is an example where simplified modelling in genome-wide association studies (GWAS) failed to find genes explaining traits that we from twin studies knew were heritable (Manolio et al., 2009), which illustrates the importance of taking the interactions between genes into account when doing genomic prediction.

Moreover, there are different levels of the genetic complexity of traits. Body mass for instance is a complex trait which is thought to be determined by many genes and is therefore referred to as a polygenetic trait. Other phenotypes, such as Huntington's disease, are explained by a single gene and are labelled as a Mendelian traits (Walker, 2007). Lastly, we have oligogenic traits which are explained by a small set of genes. The Mendelian and oligogenic traits are often categorical, while polygenetic traits are often continuous. One often observes that traditional methods of estimating breeding values have preformed worse for complex polygenetic traits than for Mendelian and oligogenic traits (Ashraf et al., 2022). These issues combined with the rise of popularity in machine learning techniques

and the increasing sample size in gene-sequencing have resulted in attempts at modelling SNPs and their relation with phenotypes using methodologies such as deep learning and tree-based methods.

As an alternative to linear statistical methods, machine learning has been shown to be able to find complex non-linear interactions in data. If polygenetic traits rely on epistatic effects, there is reasonable to assume that the gene-gene interactions can be non-linear. Machine learning methods such as trees can therefore be suitable tools for modelling polygenetic traits. Methods from machine learning impose no assumption of the distribution of the data, making them more flexible but also harder to interpret. Despite deep learning's success in other fields of science and technology (Baccouche et al., 2011; Kearnes et al., 2016; Albrecht et al., 2017; Ge et al., 2020) there has been little consistent improvement when it comes to genomic prediction (Montesinos-López et al., 2021). However, there have been findings of tree-based methods such as boosted regression trees having promising results (Gill et al., 2022). Machine learning methods are more reliant on larger datasets than traditional methods, resulting in fewer modelling attempts in the field of wild populations compared to the fields of breeding and medicine (which typically have more resources)(see Li et al., 2018; Nazzicari and Biscarini, 2022; Chafai et al., 2023). Therefore, many promising methodologies developed for exploring gene-gene and gene-environment interactions, such as the one proposed by Johnsen et al. (2021), are simply not feasible in wild population studies due to the requirement of a large dataset. This is a challenge as individuals in natural populations are more exposed to a varying environment making the gene-environment interactions more important to understand.

In this project, we investigate how one can make polygenetic phenotype predictions based on genotypes for wild populations with boosted regression trees. This is done in two ways: We first use a two-step method where we only look at the genetic component of the phenotype and try to estimate the breeding value of different individuals. As a second approach, we model gene-environment interactions to predict the phenotype directly. We compare a number of different boosting algorithms namely XGBoost (Chen and Guestrin, 2016) which has been the de-facto algorithm for tabular data in many competitions, CatBoost (Prokhorenkova et al., 2017), LightGBM (Ke et al., 2017) and gradient boosting machine (GBM) (Friedman, 2002; Pedregosa et al., 2012). We outline how to systematically perform hyperparameter tuning, and also explore some different feature selection methods.

2 Background

2.1 Traditional Methods in Ecological Genetics

Before delving into the machine learning-modelling, it is important to get an understanding of the data one works with and what it represents. It is also important to have a basic understanding of which traditional methods have been used and how they work. In this section we therefore review some traditional methods and principles that have been used by ecologists.

2.1.1 Basic Genetics

Sexually reproducing organisms inherit a set of two alleles, one from their father and one from their mother (Conner and Hartl, 2004a). These two alleles are placed on the same locus, and together they make up a single genotype. A locus refers to a location along a chromosome. The alleles can either be dominant (A) or recessive (a), thus we have three different unique combinations that can occur. We either have two recessive alleles (aa), two dominant ones (AA) or one of each (Aa), where the two first options are called homozygous effects while the latter is called hetrozygous. A way to quantify the genotypes is through marker based gene sequencing, where each genotype is classified as either 0 (recessive homozygous genotype), 1 (hetrozygous genotype) or 2 (dominant homozygous genotype). As much of the genetic material is identical across species (and thus uninteresting) it is common to only sequence the genotypes that differ between individuals which are called SNPs (single nucleotide polymorphisms).

A way to quantify genetic variation is through allele frequencies. Consider a fixed locus in a population and let q_i be the frequency of allele A, and let p be the frequency of the allele a at the same locus in the same population. That is $p = n_a/n$ and $q = n_A/n$, where n_a is the number of individuals with an a allele, n_A is the number of individuals with an A allele and n is the total number of individuals. As the allele must either be a or A, we have that $p + q = 1$.

2.1.2 Quantitative Genetics and Phenotypic Variation

A phenotype P of an *individual* is thought to be made up by effects from the environment E and effects from the inherited genes G, that is $P = E + G$ (Conner and Hartl, 2004b). One can further decompose the genetic effect G into the effects from additive genetic effects, dominance effects and epistatic effects. Additive genetic effects refer here to effects from independent contributions of alleles, dominance are effects from interactions between alleles at the same locus, while epistatic effects are from interactions between alleles at different locus. In quantitative genetics one studies quantitative traits which are believed to be made up by many genes. As genotypes in sexually reproducing organisms are not clones of their parents' genotypes, but rather a new random combination of the

parent's genes (Conner and Hartl, 2004b), the genes explaining a phenotype are thought to contribute with independent effects. Therefore, one is in particular interested in quantifying the effect from the additive part of the genotype, which is called the breeding value. The breeding value describes the individual effect a parent's genes have on the offspring's phenotype.

From $P = E + G$, it follows that the variation of a phenotype between individuals in a population is $V_P = V_E + V_G + V_R$, where V_E is the phenotypic variation due to the environment, V_G is the variation due to variation in the genome and V_R are variation due to random noise but also variation due to gene-environment effects. As genes are passed on from parent to offspring V_G gives a measure of heritability, and thus also a quantification of how fast potential selection can drive forth an evolution of the phenotype. More quantitatively, we define the *broad sense heritability* $h_B^2 = V_G/V_P$ which measure the proportion of variation in a phenotype that is due to inherited effects. One frequently breaks down the genetic variance further into the sum of the additive genetic variance V_A, the variance explained by dominant effects V_D, and the variance explained by epistatic effects V_I. The variance of the breeding values of a phenotype in a population is then V_A. To quantify the proportion of variation in a trait due to additive genetic effects we define the *narrow sense heritability* $h^2 = V_A/V_P$ (Conner and Hartl, 2004b). In the field of genomic prediction, the broad and narrow sense heritabilities give an upper limit to the prediction accuracy, and especially the narrow heritability is used when assessing the accuracy of predicted breeding values.

2.1.3 The Animal Model

One of the pioneering and widely used methods to model the phenotypic trait $\mathbf{y}$ as function of genetic and environmental variables is the animal model (C.R. Henderson, 1984; Kruuk, 2004; Wilson et al., 2010). Animal models were first based on pedigrees of the individuals and could therefore quantify genetic effects without the need for gene sequencing, which can be costly and time-consuming. In its simplest form it can be stated as a linear mixed model,

$$y_i = \mu + g_i + \varepsilon_i \quad i = 1, \ldots, n, \tag{2.1}$$

where μ denotes the population mean of a trait, g_i is the breeding value of individual i, and ε_i accounts for the environmental and non-additive effects. Here g_i is a random effect, $\mathbf{g} \sim N(\mathbf{0}, V_A\mathbf{G})$, where V_A is the additive genetic variance and $\mathbf{G} \in \mathbb{R}^{n\times n}$ is the relatedness matrix which quantifies how much the different individuals $i = 1, \ldots, n$ are related to each other, for instance if individual i and j are parent and offspring then $\mathbf{G}_{ij} = 0.5$ as they should on average share half of the genetic material. The animal model is highly flexible as one can include multiple fixed and random effects to account for specific environmental effects. This allows for investigation of the variation between different subgroups in the populations such as gender, geographic location or variation between offsprings from different mothers.

The animal model can also be extended to use sequenced genetic data by letting the elements of the relatedness matrix $\mathbf{G}_{ij}$ contain the proportion of genes shared between individual i and j. The genomic relatedness matrix can be constructed by using the first method proposed by VanRaden (2008). Let $\mathbf{M}$ be the matrix of dimension $n \times p$ consisting of p SNP-markers for n individuals, and let p_i be the second allele frequency at locus i. The rows of $\mathbf{M}$ then contains each individual's SNP-vector, while the columns contain each individual's genotype at a specific locus. First we define the zero-centered

matrix $\widetilde{\mathbf{M}} = \mathbf{M} - \mathbf{1}$, which means that the SNPs now are encoded as $-1, 0$ or 1 (instead of $0, 1, 2$). This gives an interpretation of the matrix $\widetilde{\mathbf{M}}\widetilde{\mathbf{M}}^T$, where the diagonal elements, $\text{diag}(\widetilde{\mathbf{M}}\widetilde{\mathbf{M}}^T)$, will be the l_2 norm of an individual's SNP vector. Each genotype is encoded as $-1, 0$ or 1, thus the l_2 norm will be the sum of number of -1s and 1s present, *i.e.*, the number of homozygous loci an individual has. The matrix products that make up the off-diagonal (*i.e.*, between individuals) will be 1 when they share a homozygous locus, 0 when one of the individuals has a hetrozygous locus and -1 if they have opposite homozygous locus. The off-diagonal elements are therefore the difference between the number of equal homozygous locus and the number of opposite homozygous locus, *i.e.*, the off-diagonal works as a measure of number of alleles shared between individuals. Second, VanRaden (2008) suggest defining a new matrix $\mathbf{Z}$ by subtracting $2(p_i - 0.5)$ from the columns/loci of $\widetilde{\mathbf{M}}$ in order to give rare alleles more credit than common ones. The genomic relatedness matrix $\mathbf{G}$ is then defined as

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

where the divisor $2\sum_{i=1}^{p} p_i(1-p_i)$ is equal to the sum of hetrozygous frequencies under a Hardy-Weinberg equilibrium (see Conner and Hartl, 2004a). The divisor scales $\mathbf{G}$ to take the hetrozygous frequencies into account as well, and makes the behavior of the genomic relatedness matrix more similar to the pedigree relatedness matrix. The estimate of the proportion of genomic material shared between individual k and j is given by $\mathbf{G}_{ij}/\sqrt{\mathbf{G}_{ii}\mathbf{G}_{jj}}$ (VanRaden, 2008). In that sense one can think of the genomic relatedness matrix as a covariance matrix, and to get "genomic correlations" one must scale it relative to the two individual's own "genomic variance".

2.1.4 BLUP and GBLUP

Animal models provide a way to decompose the phenotypic variation into different variance components such as additive genetic variance and environmental variance, and also quantify them (Wilson et al., 2010). However, it is often desirable to obtain estimates of the breeding values $\hat{g}_i$ often called EBV (estimated breeding values) (T. Meuwissen et al., 2016). For that purpose we use an approach called BLUP (best linear unbiased predictor) which still uses the animal model with a pedigree (C. R. Henderson, 1975). Assume that we have an animal model where the phenotype $\mathbf{y}$ is modelled as

$$\mathbf{y} = \mathbf{1}\mu + \mathbf{X}\boldsymbol{\beta} + \mathbf{g} + \mathbf{W}\mathbf{d} + \boldsymbol{\varepsilon}, \tag{2.2}$$

where μ is the population mean, $\boldsymbol{\beta}$ is the fixed effects, $\mathbf{g} \sim N(\mathbf{0}, V_A\mathbf{G})$ is the breeding values (random effect), $\mathbf{d} \sim N(\mathbf{0}, \mathbf{R})$ is the vector containing other random effects, $\mathbf{X}$, $\mathbf{W}$ are design matrices of appropriate dimensions and $\boldsymbol{\varepsilon} \sim N(\mathbf{0}, \sigma^2 I)$ is the residual term. $\mathbf{R}$ is a diagonal covariance matrix such that the random effects are independent of each other. The model can be rewritten to a more compact form by bundling the breeding values and the random effects together into a single vector $\boldsymbol{\gamma}$ with a block-diagonal indices matrix $\mathbf{U}$, and the mean is bundled together with the fixed effects into $\boldsymbol{\beta}^*$ with indices matrix $\widetilde{\mathbf{X}}$. The phenotype is then modelled as

$$\begin{aligned}\mathbf{y} = \widetilde{\mathbf{X}}\boldsymbol{\beta}^* + \mathbf{U}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim N(\mathbf{0}, I\sigma^2), \quad \boldsymbol{\gamma} \sim N\left(\mathbf{0}, \begin{pmatrix} V_A\mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{pmatrix}\right) = N(\mathbf{0}, \mathbf{C}) \\ \implies \mathbf{y} \sim N(\widetilde{\mathbf{X}}\boldsymbol{\beta}^*, \mathbf{U}\mathbf{C}\mathbf{U}^T + \sigma^2 I) = N(\widetilde{\mathbf{X}}\boldsymbol{\beta}^*, \mathbf{V}),\end{aligned}$$

which reduces to the pseudo linear regression problem if we write

$$\mathbf{y} = \widetilde{\mathbf{X}}\boldsymbol{\beta}^* + \boldsymbol{\epsilon}, \quad \epsilon = \mathbf{U}\boldsymbol{\gamma} + \boldsymbol{\varepsilon} \sim N(\mathbf{0}, \mathbf{V}),$$

which can be standardized by multiplying with $\mathbf{V}^{-1/2}$,

$$\mathbf{V}^{-1/2}\mathbf{y} = \mathbf{V}^{-1/2}\widetilde{\mathbf{X}}\boldsymbol{\beta}^* + \mathbf{V}^{-1/2}\boldsymbol{\epsilon} \iff \mathbf{y}^* = \widetilde{\mathbf{X}}^*\boldsymbol{\beta}^* + \boldsymbol{\epsilon}^*, \quad \boldsymbol{\epsilon}^* \sim N(\mathbf{0}, I),$$

which is an ordinary linear regression problem. The unbiased MLE estimate of $\boldsymbol{\beta}^*$ is then

$$\hat{\boldsymbol{\beta}}^* = \left(\widetilde{\mathbf{X}}^{*T}\widetilde{\mathbf{X}}^*\right)^{-1}\widetilde{\mathbf{X}}^{*T}\mathbf{y}^* = \left(\widetilde{\mathbf{X}}^T\mathbf{V}^{-1}\widetilde{\mathbf{X}}\right)^{-1}\widetilde{\mathbf{X}}^T\mathbf{V}^{-1}\mathbf{y}. \tag{2.3}$$

To obtain estimates of the random effects $\boldsymbol{\gamma}$ we use that for two Gaussian variables $\mathbf{z}_1$ and $\mathbf{z}_2$ on the form

$$\begin{pmatrix}\mathbf{z}_1\\ \mathbf{z}_2\end{pmatrix} \sim N\left(\begin{pmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{pmatrix}, \begin{pmatrix}\Sigma_{1,1} & \Sigma_{1,2}\\ \Sigma_{2,1} & \Sigma_{2,2}\end{pmatrix}\right),$$

we have that

$$E(\mathbf{z}_1|\mathbf{z}_2) = \boldsymbol{\mu}_1 + \Sigma_{1,2}\Sigma_{2,2}^{-1}(\mathbf{z}_2 - \boldsymbol{\mu}_2).$$

Thus, if we look at

$$\begin{pmatrix}\boldsymbol{\gamma}\\ \mathbf{y}\end{pmatrix} \sim N\left(\begin{pmatrix}\mathbf{0}\\ \widetilde{\mathbf{X}}\boldsymbol{\beta}^*\end{pmatrix}, \begin{pmatrix}\mathbf{C} & \mathbf{C}\mathbf{U}^T\\ \mathbf{U}\mathbf{C} & \boldsymbol{V}\end{pmatrix}\right),$$

we get that

$$E(\boldsymbol{\gamma}|\mathbf{y}) = \mathbf{C}\mathbf{U}^T\mathbf{V}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\boldsymbol{\beta}^*). \tag{2.4}$$

An unbiased prediction of $\boldsymbol{\gamma}$ will then be to insert (2.3) into (2.4), giving the EBV $\hat{\mathbf{g}}$ for each individual, which, when using the described approach, are called BLUPs (Henderson, 1950; Kruuk, 2004).

When replacing the pedigree matrix $\mathbf{G}$ with the genomic relatedness matrix, derived from the SNPs of each individual, we get what is called the genetic estimated breeding values (GEBV) and the predictions of the breeding values are called GBLUPs (genetic best linear unbiased predictor) (VanRaden, 2008; T. Meuwissen et al., 2016).

2.2 Bayesian Methods for Animal Models

We review some theory regarding Bayesian statistics which is needed to understand how animal models can be fitted using the integrated nested Laplace approximation (INLA) approach.

2.2.1 Bayesian Modelling

In traditional frequentist statistics one believes that the parameter θ is a fixed unknown number or vector. The parameter is estimated through the data $\mathbf{y}$, giving $\hat{\theta}$ which is a random variable as the data is modelled to be random. This is the frequentist point of view. Another paradigm is the Bayesian point of view, which is that the parameter θ is

random and follows a distribution $p(\theta)$ denoted the prior distribution (Casella and Berger, 2002). Bayesians seek a distribution of the parameter given the data we have observed, this distribution is called the posterior distribution $p(\theta|\mathbf{y})$. By using Bayes rule one can relate the posterior to the data and prior,

$$p(\theta|\mathbf{y}) = \frac{f(\mathbf{y}|\theta) \cdot p(\theta)}{m(\mathbf{y})}, \tag{2.5}$$

where $f(\mathbf{y}|\theta)$ is the likelihood (often denoted $L(\mathbf{y}, \theta)$ in traditional statistics), $p(\theta)$ is the prior distribution and $m(\mathbf{y})$ is the marginal distribution of the data given by

$$m(\mathbf{y}) = \int f(\mathbf{y}|\theta)p(\theta)d\theta.$$

The goal in Bayesian analysis is to find the posterior distribution and use it to make inference about the data. Bayesians often use Markov Chain Monte Carlo (MCMC) techniques to generate samples from the posterior distribution (Geof H. Givens and Jennifer A. Hoeting, 2013). MCMC is non-deterministic, often time-consuming and have no guarantee of reaching convergence. In recent years newer deterministic methods have arisen such as the integrated nested Laplace approximation approach (INLA) (Rue et al., 2009). INLA is explained more in detail in the next sections.

2.2.2 Hierarchical Models

A popular modelling approach within Bayesian modelling is the hierarchical (latent) model which generally consists of three parts (Gamerman and Lopes, 2006). The first part is the *observational* model of the data $\mathbf{y}$ which, when conditioned on the latent field $\mathbf{x}$ and some hyperparameters $\boldsymbol{\theta}_1$, is independent and identically distributed,

$$f(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_1) = \prod_{i=1}^{N} f(y_i|\mathbf{x}, \boldsymbol{\theta}_1),$$

the observational model thus contains information about the generation of the observed data. The second part of the hierarchical model is the *latent* model $\mathbf{x}|\boldsymbol{\theta}_2$ which contains information about the unobserved process, where $\boldsymbol{\theta}$ contains hyperparameters for both the latent model. The third part of the model is the *hyperpriors* $\pi()$ that are placed on the hyperparameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$.

As an example of a well-known model that fits the framework of Bayesian hierarchical models, consider a simple linear regression model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \dots N \quad \varepsilon_i \sim N(0, \sigma^2)$$

where (y_i, x_i) is observation i. The Bayesian way to specify this model would be to assume that,

$$y_i|\beta_0, \beta_1, \sigma^2 \sim N(\mu_i, \sigma^2), \quad \mu_i = \beta_0 + \beta_1 x_i,$$

which make up the observational part of the model. β_0 is the intercept and β_1 is the regression slope. The latent part of the model would be to assume that $\beta 0$ and β_1 follows some prior distribution, where a popular prior is

$$\beta_0 \sim N(a_0, \nu_0^2), \quad \beta_1 \sim N(a_1, \nu_1^2),$$

where a_1 are user-defined numbers. We thus have $\theta_1 = \sigma^2$ as the hyperparameter for the observational model and $\boldsymbol{\theta}_2 = \{\nu_0^2, \nu_1^2\}$ as hyperparameters for the latent model. Thus, to complete the hierarchical model we only need to place some hyperpriors on the hyperparameters $\boldsymbol{\theta} = \{\sigma^2, \nu_0^2, \nu_1^2\}$, where a popular choice is the inverse-gamma distribution. While the hierarchical model may seem too complex for a simple linear regression, it provides a powerful framework for situations where the data is nested with several layers of clusters. The hierarchical model is especially handy when building intertwined mixed models where the inference about model parameters can be difficult to interpret.

2.2.3 Bayesian Animal Model by INLA

Integrated nested Laplace approximation (INLA) is an efficient method obtain approximate posterior distributions for latent Gaussian models (Rue et al., 2009). The details of the numerical scheme of INLA are beyond the scope of this project, but the reader is referred to Rue et al. (2009) and Martino and Riebler (2019) for a throughout explanation.

A latent Gaussian model is a hierarchical model where the latent field $\mathbf{x}|\boldsymbol{\theta}_2$ is assumed to be Gaussian,

$$\mathbf{x}|\boldsymbol{\theta}_2 \sim N(0, \mathbf{Q}^{-1}(\boldsymbol{\theta}_2)),$$

where $\mathbf{Q}$ is the precision matrix, *i.e.*, the inverse of the covariance matrix which INLA requires to be sparse. It is easy to see that simple regression model considered in Section 2.2.2 is a latent Gaussian model if one put $a_0 = a_1 = 0$ in the priors of β_0 and β_1.

Animal models on the form of (2.2) can easily be stated as Gaussian latent model, and thus fitted using the INLA approach by letting $\mathbf{x} = (\boldsymbol{\beta}, \mathbf{g}, \mathbf{d})$ be the latent field and place a Gaussian prior on $\boldsymbol{\beta}$ with zero mean, $\boldsymbol{\beta} \sim N(0, I\sigma_\beta^2)$. The hyperparameters are then $\boldsymbol{\theta} = \{V_A, \sigma^2, \mathbf{R}\}$, and the observational model is

$$\mathbf{y}|\boldsymbol{\beta}, \mathbf{g}, \mathbf{d} \sim N(\boldsymbol{\eta}, \sigma^2 I), \quad \boldsymbol{\eta} = \mathbf{1}\mu + \mathbf{X}\boldsymbol{\beta} + \mathbf{g} + \mathbf{W}\mathbf{d}.$$

Note that σ_β^2 not was included in $\boldsymbol{\theta}$, which is due to that we want $\boldsymbol{\beta}$ to act like fixed effects where we don't want to model its variance as a hyperparameter. One typically chooses a prior with large variance for fixed effects, which means that the posterior distribution of the fixed effects will be dominated by the data. INLA also requires a sparse inverse precision matrix $\mathbf{Q}$, which is fulfilled for the animal model as the inverse of the relatedness matrix $\mathbf{G}$ is sparse (Steinsland and Jensen, 2010).

Using the INLA framework one ends up with approximate marginal posterior sample distributions for the latent field, $\tilde{\pi}(x_l|\mathbf{y})$, and for the hyperparameters, $\tilde{\pi}(\theta_j|\mathbf{y})$. It is thus easy to obtain GEBVs by taking the mode of the sample distribution of the breeding values $\tilde{\pi}(g_i|\mathbf{y})$. For more explanations and examples of how INLA can be used with animal models we refer the reader to Holand et al. (2013).

2.3 Modern Machine Learning Techniques

While traditional methods have demonstrated their worth in the use of genomic selection, recent advances in machine learning also have reached the field of genomics (Montesinos-López et al., 2021). In this section we review the theory behind a supervised machine

learning method called boosted regression trees. We first explain the general principles behind regression trees and boosting, before explaining a specific method called XGBoost in detail and how it compares it with other popular boosting methods.

2.3.1 Supervised Learning in General

A general supervised machine learning prediction problem is one in which some predictors (often called features or covariates) $(x^1, x^2, \dots x^p)^T = \mathbf{x}$ and a response y are modeled to be connected by a function $f(\cdot)$ (Hastie et al., 2009). The function is often flexible and contains a set of learnable parameters denoted β that are chosen by minimizing a certain loss called L. Thus, the problem can mathematically be stated as

$$\hat{y} = f(\mathbf{x}, \hat{\beta}), \quad \hat{\beta} = \arg\min_{\beta} L(y, f(\mathbf{x}, \beta)),$$

where $\hat{y}$ denotes the prediction of the true response y. A common choice of loss function is the squared error loss,

$$L(y, \hat{y}) = (y - \hat{y})^2. \tag{2.6}$$

Unlike classical statistics, the machine learning approach makes few, if any, assumptions about the distribution of the variables y and $\mathbf{x}$ (Molnar, 2022). As a result of the function $f(\mathbf{x}, \beta)$'s complexity and difficulty in understanding, many people refer to it as a "black box" in which it is near impossible to grasp how $\mathbf{x}$ and y connect to one another, such as which predictors are most crucial. This is due to the fact that traditionally machine learning has concerned itself with the problem of prediction and not inference. However, as machine learning is put in more use, the need for explanation also rises, giving birth to a new area of machine learning often called explainable AI (XAI), which creates techniques to make inference about the black-box models (Molnar, 2022).

The "learning" in machine learning comes from the model's use of a set of multiple observations called the data $D = (Y, X)$ to find the best set of parameters $\hat{\beta}$. For each observation of the response y_i, we observe p covariates denoted x_i^m, $i = 1, \dots N$, $m = 1 \dots p$. Let $X = \{\mathbf{x}_i\}_{i=1}^N$ be the matrix containing the covariate vectors $\mathbf{x}_i = \{x_i^m\}_{m=1}^p, i = 1 \dots N$ and let $Y = \{y_i\}_{i=1}^N$ be the vector containing all the observations of the response. To assess the model's general performance we often train our function on a subset of the total data called the train set $D_{\text{train}} = (Y_{\text{train}}, X_{\text{train}})$. Then we test the model on an unseen test set (not part of the training set) $D_{\text{test}} = (Y_{\text{test}}, X_{\text{test}})$ by making predictions of each of the response observations in the test set, which gives the generalization error; a measure of how well the model does it in new situations.

2.3.2 Regression Trees

A machine learning technique that has been explored in this project is regression trees (Breiman et al., 1984). Regression trees are decision trees that deal with the regression problem (Breiman et al., 1984; Hastie et al., 2009). A decision tree aims to split the predictor space into J disjoint rectangular regions $R_1, R_2, \dots R_J$. Each region R_j is equipped with a weight w_{R_j}, $j = 1, \dots J$ (see Figure 2.1). The tree then makes a new prediction for a new observation $\mathbf{x}^*$ by placing it in one of the regions R_j and then predicts y^* by using the weight of the region w_{R_j}, which often is the mean value (when using squared error

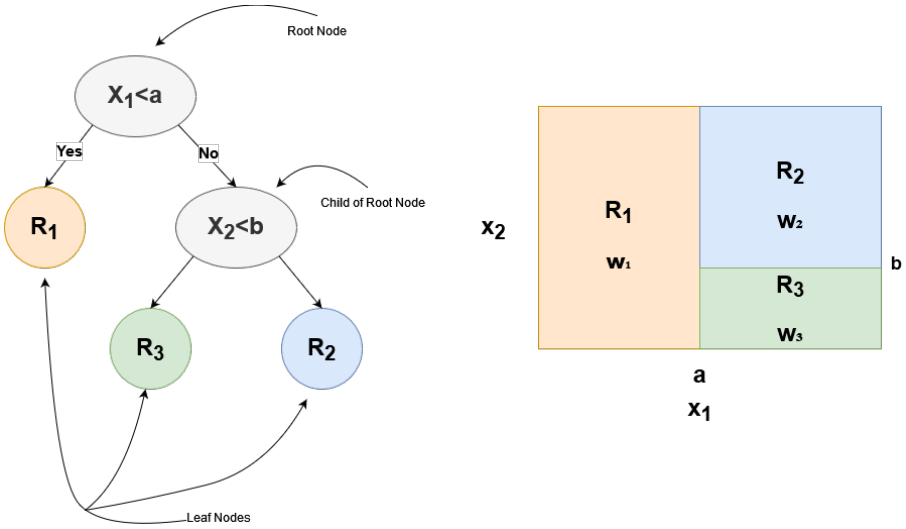

Figure 2.1: Simple decision tree with some terminology (left) and corresponding division of feature space (right)

loss) of the training observations that belong to region R_j. The tree can be described as

$$\hat{y}^* = f(\mathbf{x}^*) = \sum_{j=1}^{J} \hat{w}_{R_j} I(\mathbf{x}^* \in R_j), \qquad \hat{w}_{R_j} = \arg\min_{w_{R_j}} \sum_{\mathbf{x}_i \in R_j} L(y_i, w_{R_j}) \underset{\text{error loss}}{\overset{\text{Squared}}{=}} ave(y_i | \mathbf{x}_i \in R_j),$$

where $I()$ is the indicator function. Figure 2.1 shows a simple decision tree and its corresponding partition of the feature space. The regions are determined by minimizing the total loss over all regions,

$$\sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_j} L(y_i, w_{R_j}).$$

As it is infeasible to consider every possible combination of the regions R_j a greedy strategy called recursive binary splitting is used. Starting with putting all observations in one region, one seeks the best binary split that splits the remaining predictor space in half. That is, in each step we seek two regions $R_j(s, m)$ and $R_{j+1}(s, m)$ where s is the splitting criterion for predictor m. These regions correspond to the subspace of the predictor space given by

$$R_j(s, m) = \{X | x^m \leq s\} \quad \text{and} \quad R_{j+1}(s, m) = \{X | x^m > s\}. \tag{2.7}$$

The possible choices of s will be the available values for x^m in the training set *i.e.*, X^m_{train}. We can then quickly find the combination of m and s that give the lowest error through an iterative search. The recursive binary splitting strategy therefore provides a fast method of training trees. There are, however, some obvious downsides with regression trees. If one does not have a stopping strategy, by for instance using a validation set, trees will have a tendency to overfit the data by simply dividing all the training samples into their own separate regions. The validation set is a subset of the data that is not part of the training set or the test set, and can be used to detect overfitting by testing the model on the validation set during training. One should stop training when the validation error stops improving, as this is a clear sign of overfitting. If one does not have a validation set one often prunes the tree after training, or one places restrictions on the tree by limiting how deep it is allowed to grow. In addition, trees are not robust learners, small changes in the

training set have great effect on the outcome of the trees. Trees are also non-deterministic, often one will have multiple predictors that give the best local split leading to a random choice of possible splits, which of course may affect the model performance.

2.3.3 Boosted Regression Trees

A way to combat issues with weak learners, such as trees, is by creating an ensemble, *i.e.*, combining multiple weak learners to make a "wise crowd" (Hastie et al., 2009). One way of creating an ensemble in systematic way is through boosting (Friedman, 2001). Boosting is a general procedure that can be used to any learner. However, here we will restrict ourselves to consider boosted regression trees. In boosting one seeks to iteratively train weak learners building upon the previous one. A traditional way of boosting is to first train your weak learner with Y as response, resulting in a predictor $\hat{f}_1(x)$ and a vector of predictions $\hat{Y}_1 = \hat{f}_1(X_{\text{train}})$, and then train your next weak learner with the vector of residuals $\hat{\varepsilon}_1 = Y - \nu\hat{Y}$ as response, where ν is the learning rate and a shrinkage parameter. We then do this B times resulting in B predictors $\hat{f}_b(x), b = 1 \dots B$. The final predictor is then a regularized sum of the base predictors,

$$\hat{f}(\mathbf{x}) = \hat{f}^B(\mathbf{x}) = \sum_{b=1}^{B} \nu \hat{f}_b(\mathbf{x}), \quad \hat{f}^b(\mathbf{x}) = \hat{f}^{b-1}(\mathbf{x}) + \nu \hat{f}_b(\mathbf{x}),$$

the idea is that the residuals identify drawbacks in the previous learner and decides which gaps the next learner should try to fill. There exists many variants to the boosting fitting procedure and a special one that is used in this thesis is called gradient boosting.

The gradient of a function $F(\mathbf{x})$ gives the direction (relative to $\mathbf{x}$) in which $F()$ increases quickest, meaning that the negative gradient $-\nabla F(\mathbf{x})$ gives the direction in which $F()$ decreases the fastest (Curry, 1944). The method of gradient descent is optimization technique where the gradient is used to find the argument that minimizes a function $F()$ (Curry, 1944). Starting with an initial guess $\mathbf{x}_0$ one can iteratively minimize the function of interest by using the update rule

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \nu \nabla F(\mathbf{x}_n),$$

where ν is the step size (or learning rate). The (negative) gradient $-\nabla F(\mathbf{x}_n)$ gives the direction where $F()$ decreases the fastest when being at point $\mathbf{x}_n$, while ν determines how big step we take in that direction. Assume now that we want to find a function f that predicts y by minimizing a certain loss $L()$. The objective can be expressed by

$$\hat{f}(\mathbf{x}) = \arg\min_f L(y, f(\mathbf{x})),$$

which is minimization problem, meaning that we can apply the method of gradient descent can be applied. Thus, starting with an initial guess f^0, we can iteratively find the next best function by

$$\hat{f}^b = \hat{f}^{b-1} + \nu \nabla L(y, f^{b-1}(\mathbf{x})),$$

meaning that if the next tree $\hat{f}_b$ approximates the gradient $\nabla L(y, f^{b-1}\mathbf{x})$ well, the loss will be minimized. By using this procedure we are performing *gradient boosting*, where one identifies drawbacks and "gaps of knowledge" by using gradients instead of residuals (Friedman, 2001; Hastie et al., 2009). That is, starting with a base tree $\hat{f}^0(x)$ in each

iteration of the learners $b = 1 \dots B$ and for each training sample $i = 1 \dots N$ we find the gradient of the previous learner $\hat{f}^{b-1}$

$$g_{ib} = - \left[\frac{\partial L(y_i, f(\mathbf{x}_i)}{\partial f(\mathbf{x}_i)} \right]_{f=\hat{f}^{b-1}}. \tag{2.8}$$

We then fit a new tree $\hat{f}_b(\mathbf{x})$ using g_{ib} as response, giving the regions R_{jb}, $j = 1, \dots J_b$. Lastly for each of these regions we find the weight for each region $\hat{w}_{R_{jb}}$ by

$$\hat{w}_{R_{jb}} = \underset{w_{R_{jb}}}{\arg\min} \sum_{x_i \in R_{jb}} L(y_i, \hat{f}^{b-1}(\mathbf{x}_i) + w_{R_{jb}}).$$

The new tree is then given by $\hat{f}_b(\mathbf{x}) = \sum_{j=1}^{J_b} \hat{w}_{R_{jb}} I(\mathbf{x} \in R_{jb})$ such that the new learner is

$$\hat{f}^b(\mathbf{x}) = \hat{f}^{b-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_b} \hat{w}_{R_{jb}} I(\mathbf{x} \in R_{jb}),$$

where the final model becomes $\hat{f}(\mathbf{x}) = \hat{f}^B(\mathbf{x})$. When following the described procedure we are minimizing the loss by using what is known as functional gradient descent (Mason et al., 1999). Note that if using squared error loss, $L(y_i, f(\mathbf{x}_i)) = \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$, the gradient for each weak learner $\hat{f}^b$ is $g_{ib} = y_i - \hat{f}^b(\mathbf{x}_i) = \hat{\varepsilon}_{ib}$. Thus, we get similar results as with the traditional boosting procedure.

2.3.4 XGBoost

A special and popular implementation of gradient boosted trees is called XGBoost (Chen and Guestrin, 2016). XGBoost is a more regularized version of gradient boosting by using a regularized learning objective, meaning that the complexity of the tree is penalized. It also uses second order gradient tree boosting or extreme gradient tree boosting where one uses the Hessian as well as the gradient. Regularization, or shrinkage, aims at reducing the variance in the model and as a consequence often reduces overfitting as well (Hastie et al., 2009). We replace our ordinary loss function L with a regularized loss function

$$\mathscr{L}(Y, f(\mathbf{x})) = \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) + \Omega(f), \quad \Omega(f) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} |w_{R_j}|^2, \tag{2.9}$$

where J is the number of terminal regions (*i.e.*, the number of leaves in the tree), w_{R_j} is the weight of region j, γ and λ is regularization parameters. Thus, we now penalize the number of regions and the total size of the weights, which together makes up a regularization of the tree size. The objective for a new tree f_b is now

$$\begin{aligned}
\hat{f}_b(\mathbf{x}) &= \underset{f_b}{\arg\min}\, \mathscr{L}\left(Y, f_b(\mathbf{x}) + \hat{f}^{b-1}(\mathbf{x})\right) \\
&= \underset{f_b}{\arg\min} \sum_{i=1}^{N} L\left(y_i, f_b(\mathbf{x}_i) + \hat{f}^{b-1}(\mathbf{x}_i)\right) + \Omega(f_b).
\end{aligned}$$

For each learner f^b, $b = 1 \dots B$ and each training sample $(y_i, \mathbf{x}_i)$, $i = 1, \dots N$ we now find the gradient g_{ib} and the Hessian h_{ib},

$$g_{ib} = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f=\hat{f}^{b-1}} \qquad h_{ib} = \left[\frac{\partial^2 L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)^2} \right]_{f=\hat{f}^{b-1}}.$$

We approximate the loss $L\left(y_i, f_b(\mathbf{x}_i) + \hat{f}^{b-1}(\mathbf{x}_i)\right)$ by a second order approximation,

$$L\left(y_i, f_b(\mathbf{x}_i) + \hat{f}^{b-1}(\mathbf{x}_i)\right) \approx L\left(y_i, \hat{f}^{b-1}(\mathbf{x}_i)\right) + g_{ib} f_b(\mathbf{x}_i) + \frac{1}{2} h_{ib} (f_b(\mathbf{x}_i))^2.$$

Our objective function $\hat{f}_b$ is can then be expressed by

$$\begin{aligned}
\hat{f}_b(\mathbf{x}) &= \arg\min_{f_b} \sum_{i=1}^{N} L\left(y_i, \hat{f}^{b-1}(\mathbf{x}_i)\right) + g_{ib} f_b(\mathbf{x}_i) + \frac{1}{2} h_{ib} (f_b(\mathbf{x}_i))^2 + \Omega(f^b) \\
&= \arg\min_{f_b} \sum_{i=1}^{N} g_{ib} f_b(\mathbf{x}_i) + \frac{1}{2} h_{ib} (f_b(\mathbf{x}_i))^2 + \Omega(f^b),
\end{aligned}$$

by removing the constant terms. Note that we essentially have replaced y_i by the gradient g_{ib} and Hessian h_{ib}. This gives the approximated regularized loss function

$$\begin{aligned}
\mathscr{L}(Y, f_b(\mathbf{x})) &\approx \sum_{i=1}^{N} g_{ib} f_b(\mathbf{x}_i) + \frac{1}{2} h_{ib} (f_b(\mathbf{x}_i))^2 + \Omega(f_b) \\
&= \sum_{i=1}^{N} g_{ib} f_b(\mathbf{x}_i) + \frac{1}{2} h_{ib} (f_b(\mathbf{x}_i))^2 + \gamma J_b + \frac{1}{2} \lambda \sum_{j=1}^{J_b} |w_{R_{jb}}|^2,
\end{aligned} \tag{2.10}$$

where J_b is the number of leaf nodes in tree b. Recall that a tree is given by $f_b(\mathbf{x}_i) = \sum_{j=1}^{J_b} w_{R_{jb}} I(\mathbf{x}_i \in R_{jb})$. We can thus expand (2.10) to be at leaf level,

$$\begin{aligned}
\mathscr{L}(Y, f_b(\mathbf{x})) &\approx \sum_{i=1}^{N} g_{ib} \left(\sum_{j=1}^{J_b} w_{R_j b} I(\mathbf{x}_i \in R_{jb}) \right) + \frac{1}{2} h_{ib} \left(\sum_{j=1}^{J_b} w_{R_j b} I(\mathbf{x}_i \in R_{jb}) \right)^2 \\
&\quad + \gamma J_b + \frac{1}{2} \lambda \sum_{j=1}^{J_b} \mid w_{R_j b} \mid^2 \\
&= \sum_{j=1}^{J_b} w_{R_j b} \left(\sum_{i=1}^{N} g_{ib} I(\mathbf{x}_i \in R_{jb}) \right) + \frac{1}{2} w_{R_j b}^2 \left(\sum_{i=1}^{N} h_{ib} I(\mathbf{x}_i \in R_{jb}) + \lambda \right) + \gamma J_b.
\end{aligned} \tag{2.11}$$

Now assuming a fixed tree structure, i.e. the number of leafs J_b and the regions R_{jb} are fixed, the optimal weight of a specific region R_{jb} is

$$w^*_{R_j b} = -\frac{\sum_{i=1}^{N} g_{ib} I(\mathbf{x}_i \in R_{jb})}{\sum_{i=1}^{N} h_{ib} I(\mathbf{x}_i \in R_{jb}) + \lambda}, \tag{2.12}$$

by differentiation of (2.11) to zero. Meaning that the optimal loss for a given tree structure is

$$\mathscr{L}^*(f_b(\mathbf{x})) = -\frac{1}{2} \sum_{j=1}^{J_b} \frac{(\sum_{i=1}^{N} g_{ib} I(\mathbf{x}_i \in R_{jb}))^2}{\sum_{i=1}^{N} h_{ib} I(\mathbf{x}_i \in R_{jb}) + \lambda} + \gamma J_b, \tag{2.13}$$

by inserting (2.12) into (2.11). Equation 2.13 gives a way of measuring the quality of a given tree structure for any twice-differentiable loss function. However, as we use a greedy binary-split strategy it is more preferable to look at the reduction when splitting a leaf

node into two new regions. Assume that the leaf node we are at is R_j and we consider splitting it into R_{j+1} and R_{j+2}, the loss reduction is then

$$\begin{aligned}\mathscr{L}_{\text{split}} = \frac{1}{2}\Bigg[&\frac{(\sum_{i=1}^{N} g_{ib} I(\mathbf{x}_i \in R_{j+1,b}))^2}{\sum_{i=1}^{N} h_{ib} I(\mathbf{x}_i \in R_{j+1,b}) + \lambda} \\ &+ \frac{(\sum_{i=1}^{N} g_{ib} I(\mathbf{x}_i \in R_{j+2,b}))^2}{\sum_{i=1}^{N} h_{ib} I(\mathbf{x}_i \in R_{j+2,b}) + \lambda} - \frac{(\sum_{i=1}^{N} g_{ib} I(\mathbf{x}_i \in R_{j,b}))^2}{\sum_{i=1}^{N} h_{ib} I(\mathbf{x}_i \in R_{j,b}) + \lambda}\Bigg] - \gamma.\end{aligned} \tag{2.14}$$

We can now train trees using a greedy binary strategy by choosing the splits that gives the greatest increase in $\mathscr{L}_{\text{split}}$. This method is shown in Algorithm 1. Using Algorithm 1 one

Algorithm 1 Greedy Recursive Binary Splitting for Trees

Input: Set of features and derivatives $\{\mathbf{x}_i, g_{ib}, h_{ib}\}_{i=1}^{N}$ and parameters λ and γ

1: Initialize tree with root node $\hat{w}^*$ using (2.12)with entire feature space as region.
2: **while** tree complexity is less than some threshold **do**
3: Choose a leaf node j having region R_j
4: **for all** predictors $m = 1, \dots p$ **do**
5: Compute the reduction in loss for all split-points s_m (recall (2.7)) using (2.14).
6: **end for**
7: Choose the predictor m and split point s_m that give the greatest loss reduction, creating two new leaves $j+1$ and $j+2$.
8: **end while**

can define the fitting procedure for XGBoost shown in Algorithm 2. XGBoost implements

Algorithm 2 XGBoost Fitting Strategy

Input: Training set $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, twice differentiable loss L, learning rate ν, number of boosting iterations B, regularization parameters γ and λ.

1: Initialize constant value model, $f^0(\mathbf{x}) = \arg\min_\eta \sum_{i=1}^{n} L(y_i, \eta)$.
2: **for** $b = 1 \dots B$ **do**
3: Compute derivatives g_{ib} and h_{ib} for all $i = 1 \dots N$
4: Fit tree structure using Algorithm 1, obtaining J_b regions $\{R_{j,b}\}_{j=1}^{J_b}$
5: Fit weights $\{w_{R_j,b}\}_{j=1}^{J_b}$ for the given tree structure using (2.12), giving a new tree $f_b(\mathbf{x})$
6: Update model $f^b(\mathbf{x}) = f^{b-1}(\mathbf{x}) + \nu f_b(\mathbf{x})$
7: **end for**

Output: $f^B(\mathbf{x})$

other strategies to prevent overfitting such as subsampling. When each tree is fitted using Algorithm 1 instead of giving it all the available training data as input, one uses a subset of the data instead. The subsampling is done both observation-wise (row-wise) and feature-wise (column-wise). Subsampling gives more exposure to the different features and observations and decorrelates the different trees.

XGBoost is also implemented in a highly optimized way in terms of speed. Having both alternative approximate methods of the algorithm described in this section, and implementation for parallel fitting procedure makes it a highly efficient method. The implementation details are not discussed here, but the reader is refered to Chen and Guestrin (2016) for more details. However, there is one method worth mentioning, which is the method behind finding the best split points in Algorithm 1. Ordinary gradient boosted trees consider

all data points available in the current leaf nodes as candidates for the next split, giving a time complexity of O(Number of features $\times$ Number of unique samples). The complexity especially becomes large when having continuous features. XGBoost instead uses a histogram-based algorithm which places continuous features into discrete bins. The different bins are then considered as split points candidates, giving far fewer candidates than when considering all the unique samples.

2.3.5 CatBoost

CatBoost is boosting algorithm developed as an alternative to XGBoost (Prokhorenkova et al., 2017). CatBoost's main selling point was that it was able to natively handle categorical features, something XGBoost at the time did not. CatBoost uses similar learning objective and fitting strategy as XGBoost, but CatBoost uses only a first order approximation, meaning that only the gradient is calculated and used, while XGBoost also use the Hessian.

However, CatBoost's main difference from XGBoost and other boosting algorithms is that CatBoost uses what is called *ordered boosting*. For ordinary gradient boosting, in each boosting iteration b we calculate the gradient of the previous learner g_{ib} (2.8) using training sample $\mathbf{x}_i$. The previous learner was trained using $\mathbf{x}_i$, meaning that the target of the next tree is directly related with the training data that the next tree is going to be constructed on. This creates a relationship between the features and the target variable that not originally was there, and as the tree will try to capture this relationship, the gradients of the next learner will be shifted compared to gradients in the test set. The shift in distribution of gradients is called the prediction shift problem and introduces bias in the boosting model (Prokhorenkova et al., 2017).

CatBoost's solution to the prediction shift problem is ordered boosting, which essentially ensures that the training of the next tree is done by using a subset of the training data, and then the gradient is calculated using a different non-overlapping subset. The tree fitting algorithm may be perceived as complicated and hard to understand, but the important take-away is that CatBoost builds the tree as the data were coming in *sequentially*, one observation at the time, see Figure 2.2.

At the start, before any training is performed CatBoost makes s independent random permutations $\{\sigma_r\}_{r=1}^{s}$ of the training data. $\sigma_r(i)$ is the mapping between an observation $\mathbf{x}_i$ original placement in the data and its position in permutation r. If the original data consisted of indexes $\{1, 2, 3\}$ and the rth permutation is $\{2, 3, 1\}$, then $\sigma_r(1) = 3$. In each boosting iteration b a random choice σ_r of the permutations is used to estimate the leaf weights. As mentioned, each tree in CatBoost treats the data as it were sequentially receiving it, one observation at the time. Therefore, during the construction of a model in boosting iteration b, CatBoost saves the temporary models $f_{r,k}^{b}$, where $f_{r,k}^{b}(\mathbf{x}_i)$ is the *current* prediction of the i-th observation using the k first observations in permutation σ_r. Similarly, $g_{b,r,k,i}$ is the gradient based on $f_{r,k}^{b}(\mathbf{x}_i)$. CatBoost construct the architecture of the tree using all available training data, but the weights of the leafs are calculated sequentially for each training observation. Let $\hat{w}_{b,r}(\mathbf{x}_i)$ be the current leaf weight for observation $\mathbf{x}_i$ based on the $\sigma_r(i) - 1$ previous observations in the permuted dataset, meaning that it is constructed using the gradients $\{g_{b,r,t-1,t}\}_{t=1}^{\sigma_r(i)-1}$. The weight can be related to the notation we established in Section 2.3.2 as $\hat{w}_{b,r}(\mathbf{x}_i) = \hat{w}_{R_jb} \iff \mathbf{x}_i \in R_{jb}$. The weight $\hat{w}_{b,r}(\mathbf{x}_i)$, which is the prediction of training observation $\mathbf{x}_i$ in boosting iteration b is then calculated by gradients that are not based on sample $\mathbf{x}_i$, which combats

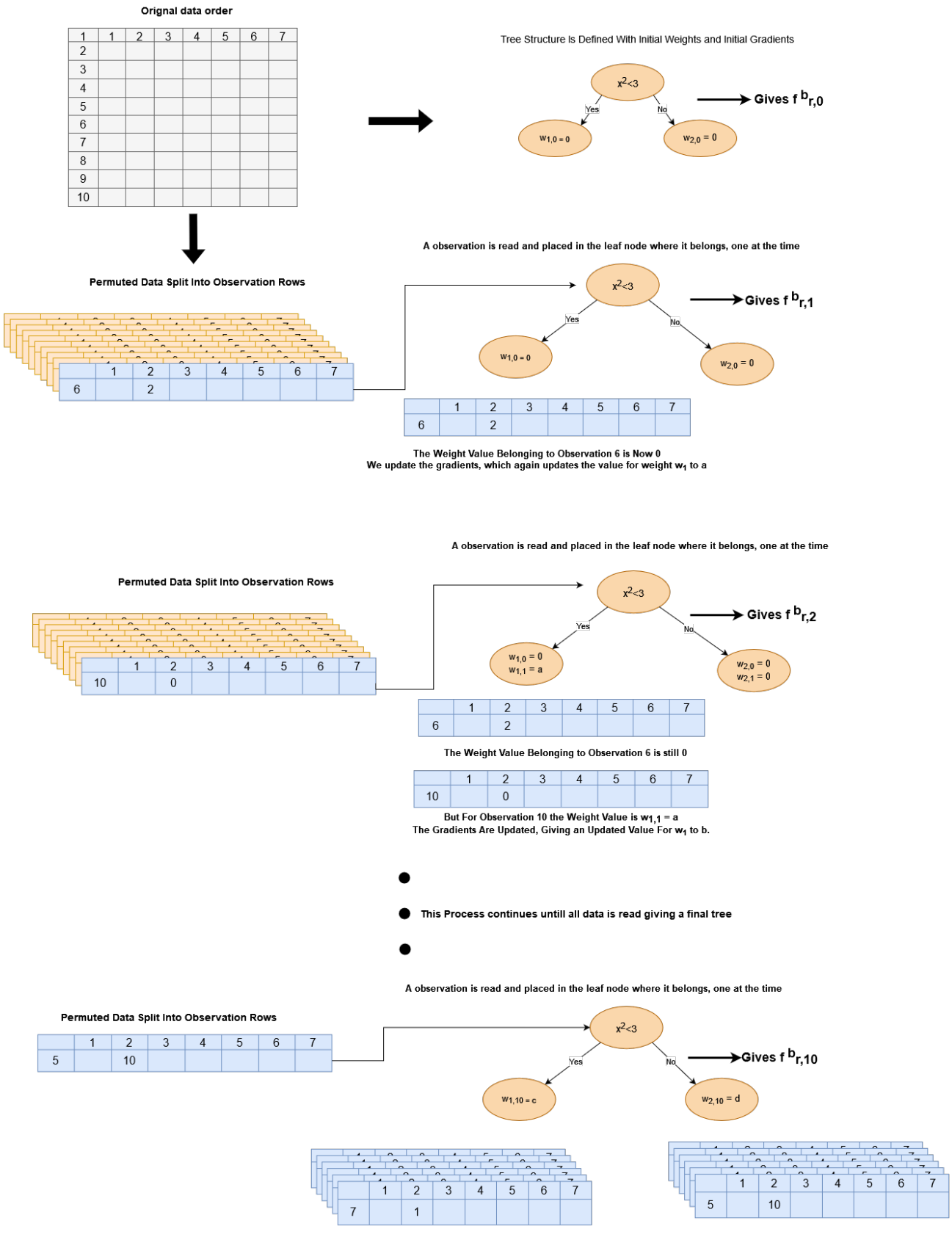

Figure 2.2: Sketch of how CatBoost builds a tree using Ordered Boosting. The data is read sequentially, and each observation is assigned its own weight by sequentially update the gradients.

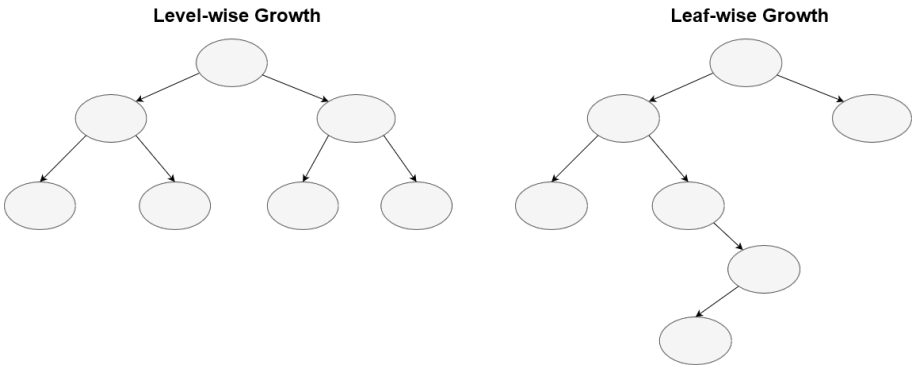

Figure 2.3: Different growth strategies for trees. (left) Level-wise growth where nodes closest to the root are prioritized. (right) Leaf-wise growth where splits creating the greatest reduction in loss are prioritized.

the prediction shift problem (Prokhorenkova et al., 2017). If one had used the same permutation of the data in each boosting iteration, the weight of the first observation $\hat{w}_{b,r}(\mathbf{x}_0)$ would always have no-previous gradients to be based, and thus be constant equal to its initial value, which is the reason why CatBoost changes between the s random permutations.

2.3.6 LightGBM

Another popular boosting method is LightGBM developed by Microsoft (Ke et al., 2017). LightGBM was initially developed as a fast and more computationally efficient alternative to ordinary gradient boosting. As CatBoost, LightGBM is a first order method that only uses the gradients. XGBoost and CatBoost grow their trees level-wise, meaning that leaf-nodes closest to the root node is considered for splitting (Figure 2.3, left). This forces a symmetric tree structure, which acts as an extra regularization of the model. LightGBM on the contrary grows trees leaf-wise, meaning that the splitting of leaf nodes is done with regard to the greatest reduction in loss (regardless of their distance to the root node), creating non-symmetric trees (Figure 2.3, right). Leaf-wise growth has the potential of increasing the accuracy at the cost of risking overfitting. LightGBM uses row-wise subsampling as XGBoost, but with a slightly different tweak to it. Samples that have larger gradients (in absolute value) contribute more to the information gain. Therefore, when performing row-wise subsampling it is wise to always keep the samples with large gradients present, ensuring that the samples that contribute the most to the information gain always is present when building the tree. This method is refered to as *gradient-based one-side sampling*.

While row-wise subsampling reduces the sample space, column-wise subsampling reduces the feature space. LightGBM implements a new method to reduce the feature space called *exclusive feature bundling* (EFB). EFB is a method which bundles features together with near-zero loss in information. By creating the bundles of the features, time complexity of finding split points in the histogram based approach reduces to $O(\text{Number of bundles} \times \text{Number of bins})$. This increases the time efficiency during training, and by grouping features together clearer differentiation between features is created as well. The bundling is done by exploiting sparsity in the data and the fact that many features rarely are non-zero at the same observation, resulting in that one can safely bundle and merge them together. We say that two features are mutually exclusive if they never take nonzero

values simultaneously.

The bundles are found by formulating the problem as a graph-coloring problem, see Figure 2.4 for some terminology related to graphs. The graph-coloring problem is a problem within graph-theory where, given a set of colors, one seeks a coloring of the nodes such that no node sharing the same edge have the same color (see Figure 2.5) (Brooks, 1941). In the EFB a graph is constructed where each node is a feature and an edge is added between two nodes if the two features are not mutually exclusive. The features that belong to the same bundle is then corresponding to the edges that are of same color.

In practice there will be few features that are completely mutually exclusive, and a slightly different graph is constructed. The features are still the nodes, but we now use weighted edges where the weight correspond to the number of mutual conflicts between two features, *i.e.*, the number of observations where both of the features are non-zero. One find the features that can be bundled together by first sorting the graph by the weight of the edges in descending order, and then traverse through the features. Starting with a set of bundles consisting of just one bundle, we iterate through the features, and for each feature we iterate through the set of bundles. For each of the bundles one finds the sum of the number of conflicts the feature have with the features in the bundle. The feature is then added to the bundled with the lowest sum of number of conflicts. However, if the sum of number of conflicts is larger than some threshold, a new bundle is created and the feature is added to this empty bundle.

Now that we have found the bundle each feature belong to, the next step of EFB to merge the features in the bundles together. EFB extends the histogram algorithm further, recall that features are placed in specific bins and the bins are used for evaluating the split-points. Each of the features in a bundle therefore have their own set of bins, EFB then first looks at the first bin for all the features, and then adds different offsets to all the first bins such that the bins are disjoint. The first bin of the bundle is then the range from the smallest offset-ed first bin to the largest offset-ed first bin. The procedure is then repeated for all the next bins of the features, creating a single set of bins for the bundle. This may seem a bit cryptic, but consider the example presented by Ke et al. (2017), where we have bundle consisting of feature A whose first bin is $[0, 15)$ and feature B whose first bin is $[10, 20]$. If we now add an offset of 10 to the bin of feature B, the two bins are disjoint and the merged bin is $[0, 30]$ which becomes the bundle's first bin.

2.3.7 General Hyperparameters in Boosting Algorithms

All boosting methods are equipped with hyperparameters which are user-defined parameters that determine much of the models overall behavior during training. Hyperparameters make the model more flexible to different use cases and when tuned correctly hyperparameters can improve the model performance. The tree based boosting methods have a wide range of hyperparameters that can be tuned and to fully understand each of the hyperparameters function, one must confer with the documentation and the reference paper of each specific method. However, the most important hyperparameters and their purpose are almost equal across all boosting method. We therefore review the most important and general groups of hyperparameters for this project.

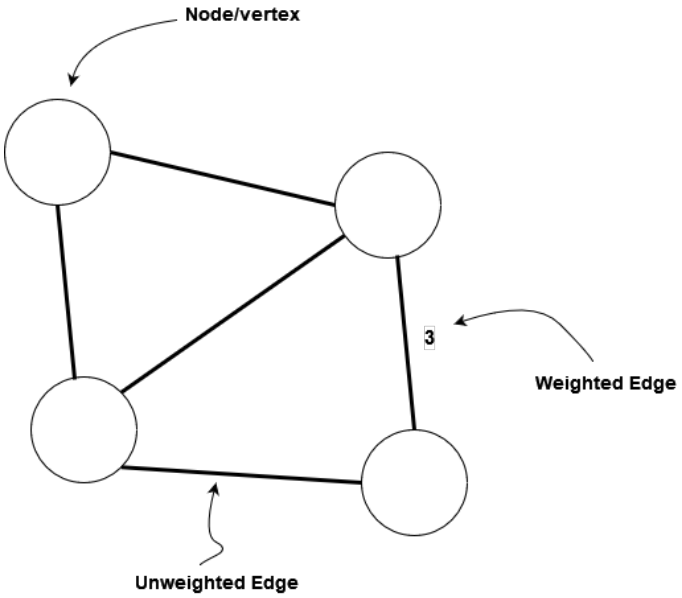

Figure 2.4: A simple graph consisting of four nodes (circles) and five edges (black lines).

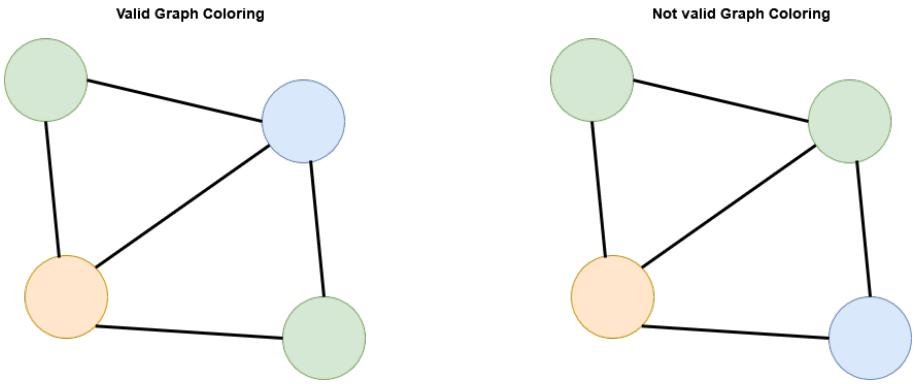

Figure 2.5: Two simple graphs illustrating the graph coloring problem. (left) A valid colored graph where no nodes that share edge has the same color. (right) Not a valid colored graph as the two green nodes share edge.

Subsampling

As mentioned in Section 2.3.4, subsampling is a technique that exploits different parts of the training observations by letting the different trees grow using different subsets of the training data. Subsampling prevent overfitting and decorrelates the trees (Friedman, 2002). The subsampling can be done observation-wise or feature-wise, where observation-wise subsampling omits entire observations $(y_i, \mathbf{x}_i)$ in the training data, while feature wise subsampling drops a random set of feature-vectors $\mathbf{x}^m$. The hyperparameters related to subsampling controls how often a new subsampling should happen and how large the subsampled subset should be. The parameter that controls how often new subsampling happens is typically called "subample_freq" or "bagging_freq", while the parameters that controls the size of the sampled subset are often called "subsample" and "colsample_bytree" for row-wise and feature-wise subsampling, respectively.

Regularization on Weights

In (2.9) we imposed an l_2 regularization on the weights through the λ parameter which is a hyperparameter that must be defined prior to training. It is also possible to add an l_1 penalty as well, which is controlled by a parameter often called α. Both l_1 and l_2 regularization combats overfitting, but they affect the weights differently. l_1 regularization has the property of putting weights to zero, while l_2 regularization has the property of shrinking correlated weights equally (Zou and Hastie, 2005).

Regularization on Tree Complexity

Similar to how the weights can be regularized, one can also impose restrictions on the complexity of each tree. As mentioned, if not regularized, trees will quickly overfit the data. One can explicitly control the tree complexity by imposing restrictions on the maximum depth of each tree or the maximum number of leaves in each tree. There are also ways to regulate the complexity more implicitly such as by requiring a certain threshold in loss-reduction (2.14) in order to split a leaf node, or one can set a requirement on the sum of weights in the new potential child nodes before splitting a leaf-node. There is also popular to set a requirement on the number of samples that belong to a new leaf node before splitting. Note that for models using leaf-wise growth strategies (such as LightGBM) to be effective, they should have a looser restriction on the maximum depth compared to the models using level-wise growth. If models having leaf-wise growth is too restricted on the depth they end up working just as a model using depth-wise tree.

Boosting Parameters

The last family of parameters worth mentioning is hyperparameters that are not tree-related but affect the boosting procedure, such as the learning rate and the number of boosting iterations. The number of boosting iterations control how many trees are fitted, and the more trees are fitted the more of the training loss is minimized. Boosting algorithms are minimizing the loss by gradient descent (Mason et al., 1999) and the learning rate (in this project called ν) controls the size of the descending steps. Thus, a high learning rate will decrease the number of iterations needed until convergence, but if the learning rate is too high one risk "missing" the minima of the loss. A low learning rate will

need more boosting iterations to reach convergence and does not risk missing minimas, but can be in danger of getting stuck in local minimas.

3 Methods

3.1 Data Description

The genetic and environmental data used in this project is from a long-term study of a meta-population of house sparrows off the Helgeland coast in Norway (Jensen, Sæther et al., 2004; Jensen, Steinsland et al., 2008; Lundregan et al., 2018). The meta-population consists of several subpopulations on eight different main islands, each with its own ecological habitat. The islands have varying population sizes and varying levels of genetic flow. Therefore, the five main islands closest to the mainland (which has the largest and most stable populations) are denoted as *inner*, the three main islands further away from the mainland is denoted *outer*, and other smaller islands are denoted *other* (Muff et al., 2019). The genetic data consists of 182 854 quality controlled SNPs, encoded as $0, 1$ or 2 depending on whether the SNP is recessive homozygous, heterozygous or dominant homozygous. The data about the environment is available in the form of the *current island* of recording and the *month* of recording. The phenotypic target variables that are modelled in this project are body mass and tarsus length. Other individual-specific data that is available and used is the *sex* of each individual, the *FGRM* (genomic individual inbreeding coefficient), *outer* (proportion of genetic material from outer island group), *other* (proportion of genetic material from other island group) and the individual's *age*. Since the data is from a long-term study conducted since 1993 there are repeated measurements of the same individual. There are 4371 body-mass recordings of 1971 unique individuals and 4495 tarsus-length recordings of 1966 unique individuals.

3.2 Genomic Prediction using Boosted-Regression Trees

We here explain how genetic prediction of the phenotypes body mass and tarsus leg length is done. The models we fit using the Helgeland house sparrow data are XGBoost, CatBoost, LightGBM and GBM using body mass and tarsus length as the target variables. All models are fitted in Python version 3.8. XGBoost models are fitted using the XGBoost Python interface of version 1.7.3, which is the latest stable version, documentation can be found at xgboost.readthedocs.io. CatBoost models are fitted using the python interface of version 1.2, documented at catboost.ai. LightGBM models is fitted using the LightGBM package of version 3.3.5 documented at lightgbm.readthedocs.io. For GBM we use the implementation provided by scikit-learn version 1.3.0 (Pedregosa et al., 2012). We also use the GEBVs and predicted phenotypes from a genomic bayesian animal model as a benchmark to compare with our new models. The animal model is fitted using the package R-INLA of version 23.09.09 with R of version 4.3.2. R-INLA is an implementation of INLA and is documented at r-inla.org.

We use two different fitting strategies creating two different responses. First we use a two-step strategy, where the genetic effects are separated from the environmental effects by first fitting a linear mixed model. The output from the mixed-model is then used as

input in the tree-based methods to predict the genetic effects. As a second approach we use a one-step procedure where both environmental and genetic effects is used as input. Regardless of the response, the model fitting procedure is the same. All code used in this project is available at the GitHub repository didrik1812/ProjectThesis.

3.2.1 Model Fitting

The models are fitted in a 10-fold cross-validation procedure, yielding a sample distribution of each model's performance, making a more representative image of the quality of the models. In the cross-validation all data is split into 10 equal parts denoted folds. One then loops over all the folds, where in each loop one fold set aside as the test set, and the other folds make up the test and validation set. As we have 10 folds, the loop repeats the procedure 10 times, each time with a new fold as test set.

In each loop, the other folds that are not used for the test set are split into a validation set and a train set. We then preform feature selection using the validation and train set. The different ways of doing feature selection are described in Section 3.4, but in any case we remain with a subset of the total features. Hyperparameter tuning is then preformed using Bayesian optimization. Bayesian optimization is described in Section 3.3.2, but as for now think of it as a procedure that outputs some suggested new hyperparameters. The train set is used to fit the model with the suggested hyperparameters, while the validation set assess its performance using a squared error loss as given in (2.6). The hyperparameter procedure is repeated 30 times, resulting in a final set of hyperparameters. With the final set of hyperparameters, we fit the model using both the validation and train set, before assessing its performance on the test set, this is similar to the experimental set up used by Prokhorenkova et al. (2017). The models are fitted and tuned using squared error as a loss function, but the final assessment is done by the Pearson correlation (3.1) between the true response and the predicted response, which is the standard in genomic prediction. In each fold we thus get an assessment of the model's performance, meaning that in total after doing all folds we are left with 10 assessments of the model's performance. These 10 scores gives a sample distribution of the model's overall performance. See Figure 3.1 for a conceptual drawing of the process.

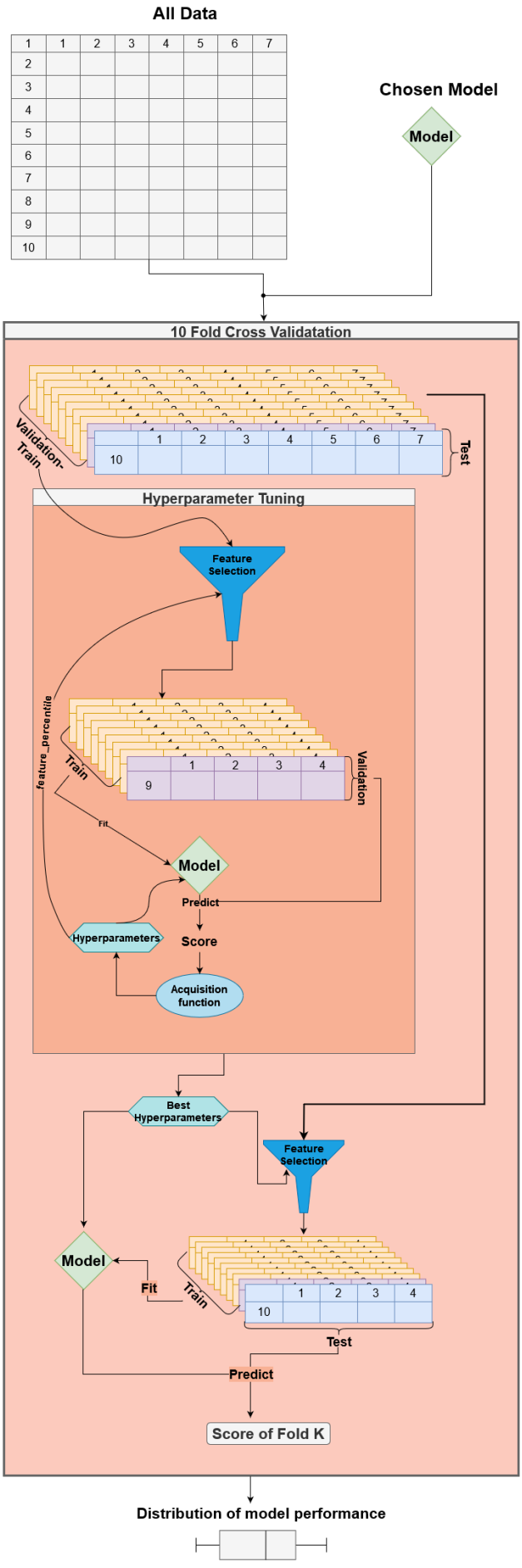
All Data
Chosen Model
Model
10 Fold Cross Validatation
Validation-Train
Test
Hyperparameter Tuning
Feature Selection
feature_percentile
Train
Validation
Fit
Model
Predict
Hyperparameters
Score
Acquisition function
Best Hyperparameters
Feature Selection
Model
Fit
Train
Test
Predict
Score of Fold K
Distribution of model performance

Figure 3.1 *(previous page)*: Conceptual flow of the cross-validation procedure for an arbitrary model and data. The 10-fold cross validation take a dataset and a model as input. The data is then split into 10 parts, where one part is set aside as test set (marked in light blue). The 9 other parts are sent to hyperparameter tuning. One of the 9 parts are set aside as a validation set (marked in purple), the other 8 parts are used to train model with new hyperparameters suggested by the Bayesian optimization. The validation set is used to measure the quality of new hyperparameters. The hyperparameter tuning outputs the set of hyperparameters that gave the highest model performance. The model is trained using the obtained set of hyperparameters and both the validation set and the train set, before its performance is assessed on the test set, yielding a performance score. The described procedure is repeated 10 times, using a different part of the data as test set each time. Output is 10 scores, which acts as a sample distribution of the model performance.

3.2.2 Two-Step Procedure

We first only try to estimate the breeding value of each individual from the genomic (SNP) data. To this end, we use a two-step procedure where we first fit a linear mixed model with the phenotype of interest as response and some environmental variables as predictors. Mathematically, this is can be expressed as

$$\mathbf{y} = \mathbf{1}\mu + X\boldsymbol{\beta} + Z\boldsymbol{\gamma},$$

where $\mathbf{y}$ is the phenotype observations of individuals and μ is the corresponding phenotype mean across all individuals. $X\boldsymbol{\beta}$ contain the fixed effects, namely the *sex* of each individual, *FGRM*, the genomic inbreeding coefficient of each individual, the *month* of observation, the *age* of each individual at time of observation, *outer* and *other* which describes how much of the genetic material in an individual origin from the *outer* or *other* island group. $Z\boldsymbol{\gamma}$ contains three random intercepts on the variables *island_current*, *hatchyear* and *ringnr* which account for potential clusters due to repeated measurements at island level, individual level and hatchyear level. Note that this model is essentially the same as (2.2), only that the breeding value is omitted. The random intercept of *ringnr* denoted $\gamma_i, i = 1, \dots n$ accounts then for individual-specific effects on the phenotype, i.e. the individual-specific genetic effects. We then define a new pseduo-response from this model, as

$$y_i^* = \hat{\gamma}_i \quad i = 1, \dots n \text{ (Assuming a total of } n \text{ individuals).}$$

This new response contains the individual-specific effect on the respective phenotype, meaning that we have separated the environmental variation from the individual-genetic variation. We can then model the genetic contribution and its relationship with specific genes using $\hat{y}_i^*$ as the response and the SNP markers as predictors.

3.2.3 One-Step Procedure

Another procedure we use is to incorporate the environmental effects directly by using the variables *age*, *month*, *other*, *island_current*, *sex*, *outer* and *hatchyear* as predictors as well as the SNP-markers, the response is now the phenotype $\mathbf{y}$. This approach allows for modelling interactions between environment and genes, as opposed to the two-step procedure and the animal model. If gene-environment interaction are present, they may affect the phenotype. However, without modelling the interactions, it is hard to determine whether they are present and what effect they have. It is therefore important to include the interaction, both in terms of prediction accuracy and for inference purposes. The one-step procedure also allows for cleaner model interpretation, in the sense that it is easier to compare the total effects of environmental variables to the total effects of genetic variables. comparison of the genetic effect and environmental effect allows us to assess whether it is the environment or the genetic material which influences the phenotype the most.

However, as we have multiple recordings of the same individual, we risk overfitting the model to recognize each individual and thus getting artificially high results. Therefore, we make the cross validation folds based on the *ringnr* variable instead of doing random splits, ensuring that the same individual is not present in both the train-set and the test-set.

As a reference to compare the boosting models, an animal model is fitted for both bodymass and tarsus-length. For the two-step procedure the GEBVs from the animal model is used as benchmark performance, while for the one-step procedure the predicted phenotype value from the animal model is used as benchmark. We fit the animal model by using a Bayesian approach with INLA as described in section 2.2.3. We use the same environmental variables as in the linear mixed model in the two-step procedure (both fixed and random), but we include the breeding values as genetic effects using a genomic relatedness matrix $\mathbf{G}$. The genomic relatedness matrix is constructed from the SNP data using the first method proposed by VanRaden (2008), which is described in Section 2.1.3.

3.3 Hyperparameter Tuning

Hyperparameters are user-defined parameters of the model, meaning that they are not directly learned from the data. However, hyperparameters affect the quality of the model greatly, and it is there important to tune them properly in order to ensure that the model is preforming at its optimal potential. Gradient-boosted tree methods typically have a vast number of hyperparameters that should be explored like the learning rate, the number of boosting iterations and tree depth.

3.3.1 General Hyperparamter Tuning

In hyperparameter tuning one wishes to find a balance between exploitation (investigate promising combinations of hyperparameters), exploration (discovering new combinations of hyperparameters) and time spent. The two most popular (and simplest) methods of hyperparameter tuning are called grid search and randomized search (Bergstra and Bengio, 2012). In grid search, a high-dimensional grid of hyperparameter values is defined, and then the model is trained and tested for all points in the grid. Randomized search on the contrary uses a random subset of the grid. While grid search has high exploitative and exploratory power, it quickly becomes time-consuming as the grid increases (both in

terms of dimensions and density of the grid). Randomized search however can be much faster but has poor exploitative power.

3.3.2 Bayesian Optimization

Let $\mathbf{x}$ be a set of hyperparameters and $F(\mathbf{x})$ be the model performance as a function of the hyperparameters. Assume that one start doing hyperparameter tuning with randomized search using a small random subset $\{\mathbf{x}_i\}_{i=1}^n$ of the entire grid, giving model performances $\{F_i\}_{i=1}^n$. It is now desirable to determine which new grid point will give the highest probability of improvement of the model performance, given the results from the randomized search. However, there are regions of the grid which are unexplored and therefore the uncertainty of the model performance in those regions are high. We thus want to maximize the probability of improvement while minimizing the uncertainty of the performance, this is essentially the exploitation-exploration dilemma. The hyperparameter tuning problem is now an optimization problem regarding the model performance.

While the problem is easy to state, it is hard to solve. How do we find the probability of improvement, how do we assess the uncertainty, and how to optimize it? The model performance as function of hyperparameters is unknown, making it hard to optimize. All we have are some data points relating model performance to model hyperparameters. One solution is called Bayesian optimization, let the model performance as a function of the hyperparameters $\mathbf{x}$ be denoted $F(\mathbf{x})$. In the Bayesian regime, similarly to how the parameters in Section 2.2.1 was treated as random variable, F is treated as a random function. Treating F as a random function lets us make a first guess on how it is expressed, which is expressed through the prior distribution $p(F)$, and then as we gather data $H = \{\mathbf{x}_i, F_n\}_{i=1}^n$ (*i.e.*, observations on sets of hyperparameters and model performance) we update our view of F by calculating the posterior distribution. In order to use the posterior to find a new set of hyperparameters, given the data we have, we use an *acquisition function*. The acquisition function is a function determined by the posterior that can be maximized to find the next point $\mathbf{x}_{n+1}$. Once the acquisition function is calculated and maximized we find the new suggested point $\mathbf{x}_{n+1}$ and train the model with it, giving a new point of performance F_{n+1}. $\mathbf{x}_{n+1}$ and F_{n+1} is then added to H and the entire process is repeated. Pseudocode for the Bayesian optimization process is given in Algorithm 3.

Algorithm 3 Bayesian Optimization

Input: A model M, a number of optimization trials N, and number of random starting points n

Place prior on F
Make a random start grid of hyperparameters $\{\mathbf{x}_i\}_{i=1}^n$
Train model M for all hyperparameters points giving scores $\{\mathbf{F}_i\}_{i=1}^n$
Collect scores and hyperparameter in $H = \{\mathbf{x}_i, F_n\}_{i=1}^n$
for $k = 1, \ldots, N$ **do**
    Calculate posterior of F using H.
    Find $\mathbf{x}_{n+k}$ by maximization of the acquisition function.
    Train M using $\mathbf{x}_{n+k}$ and obtain score F_{n+k}.
    Add $\mathbf{x}_{n+k}$ and F_{n+k} to H.
end for

Output: The best performing set of hyperparameter $\mathbf{x}^*$
    where $F(\mathbf{x}^*) = \max_{F_i \in H} F_i$.

The acquisition function may be the probability of improvement, or the width of the credible interval, but a common choice is the (posterior) expected improvement which according to the Bayesian decision theoretic framework balances bias and variance. Let $F^* = \max_{F_i \in H} F_i$ be the current best model performance, and assume that we have obtained n model evaluations. The expected improvement for a set of hyperparameter $\mathbf{x}$ is then defined as

$$EI_n(\mathbf{x}) = E_n\left([F(\mathbf{x}) - F^*]^+\right), \quad \text{where} \quad [F(\mathbf{x}) - F^*]^+ = \max(F(\mathbf{x}) - F^*, 0),$$

the next best point is then

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}} EI_n(\mathbf{x}).$$

The specific implementation details of Bayesian optimization are beyond the scope of this project, but the reader is referred to Frazier (2018) and Bergstra, Bardenet et al. (2011) for a review. To sum it up, Bayesian optimization is an iterative technique where the search space is gradually increased in size or in resolution, see Figure 3.2. We use the Python package *hyperopt* (Bergstra, Yamins et al., n.d.) to implement Bayesian optimization using the tree-structured parzen estimator approach (TPE) (Bergstra, Bardenet et al., 2011) in our model fitting-procedure. The TPE algorithm is not described here, but it builds on the ideas of Bayesian optimization (see Bergstra, Bardenet et al., 2011).

For XGBoost we tune the parameters listed in Table 3.1 for 30 Bayesian optimization trials. The parameters we choose to tune is the same as Prokhorenkova et al. (2017) uses in their experimental set up. Similar parameters are tuned for CatBoost, LightGBM and GBM, see Section A.

3.4 Feature Selection

We are dealing with numerous SNPs, where most of them will be completely unrelated to the phenotype of interest. In order to keep computational costs (time and memory) manageable it is desirable to perform feature selection before training in order to reduce the feature space. Feature selection will also help the model focus on important features and its interactions. We test three different methods of doing feature selection.

1. **Pearson correlation**: select the k SNPs most correlated with the response (in the train set) using Pearson's correlation coefficient. It has the property of finding linear relationships between the features and the response, and is defined as

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{3.1}$$

   assuming we have n samples in the training set.

2. **Spearman correlation**: again select the k most correlated SNPs with the response, this time using Spearman's rank correlation coefficient. Spearman's correlation has the property of describing a monotonic relationship, meaning that the relationship can be non-linear. Instead of using the variable values directly, it uses the rank of x and y, denoted $R(x)$ and $R(y)$, where $R(x_i) = j \implies x_i = x_{(j)}$, meaning that x_i is the jth largest value. The coefficient is then calculated using Pearson's correlation as

$$\rho_{xy} = r_{R(x)R(y)}.$$

Table 3.1: Hyperparameter Distributions for XGBoost. The *Name* column contain the name of the hyperparameter that is tuned. The *Description* column contains a short description of the hyperparameter that is tuned (see Section 2.3.7 for a general description of the hyperparameters). The *Distribution* column contains the distribution of potential values the hyperparameter can take, it establishes the search grid of the hyperparameters.

Name	**Description**	**Distribution**
max_depth	Maximum depth of a tree	Random integers from $[2, 10]$
alpha	L1 regularization parameter	Log-uniform at $[e^{-8}, e^{2}]$
lambda	L2 regularization parameter	Log-uniform at $[e^{-8}, e^{2}]$
min_child_weight	Minimum sum of instance weight needed in a child node to split	Log-uniform at $[e^{-8}, e^{5}]$
eta	The learning rate	Log-uniform at $[e^{-7}, 1]$
subsample	The fraction of training data used for each tree	Uniform at $[0.5, 1]$
n_estimators	The number of trees used	Random integers from $[20, 205]$
colsample_bytree	The fraction of columns used for each tree	Uniform at $[0.5, 1]$
gamma	The threshold in loss reduction (Equation 2.14) required to split a node	Log-uniform at $[e^{-8}, e^{2}]$
feature_percentile	Fraction of SNPs used in booster	Uniform at $[0.1, 0.9]$

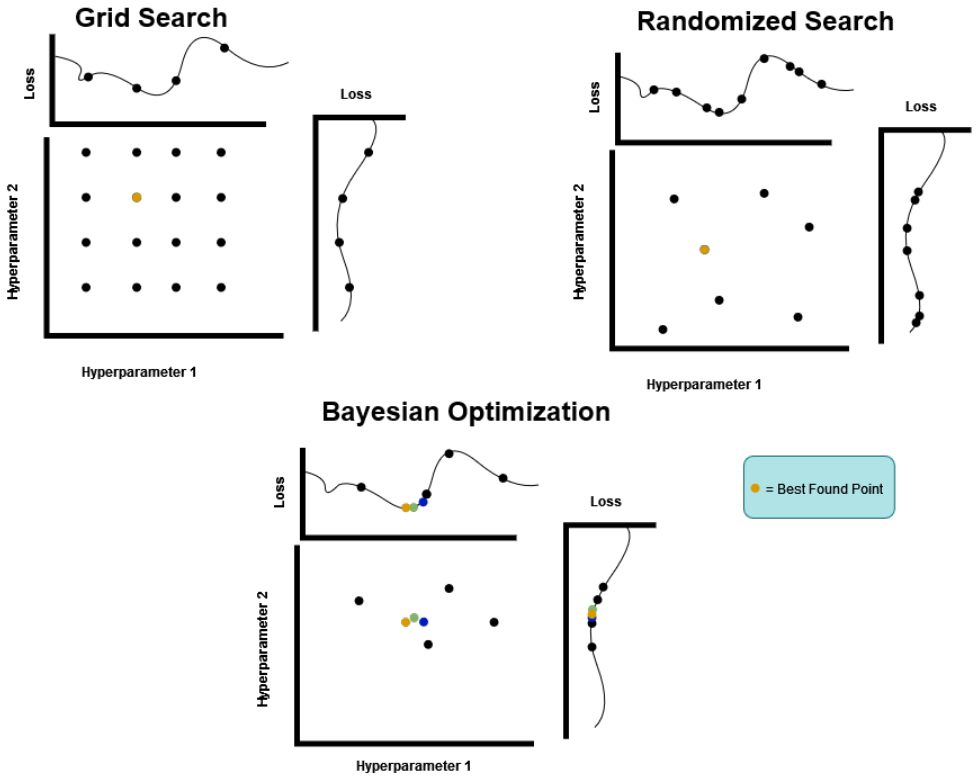

Figure 3.2: Sketch of different hyperparameter strategies. Black dots show grid points of the hyperparameter space. The loss curve evaluates the different grid points. Grid search creates a $n \times n$ grid with n unique values for hyperparameter 1 & 2. Randomized search creates a random grid consisting of random values for hyperparameter 1 & 2 drawn from some pre-specified distribution. Bayesian optimization starts with a random grid, and from there on explore the most promising values. The yellow dots show the best found point for each strategy. For Bayesian optimization, the black dots are the random starting set, while the colored points are suggested new points.

3. **Kendall correlation**: same procedure as the in the two previous ones, but use now Kendall's rank correlation coefficient (Kendall, 1945). As the SNPs are encoded to only take three values, we will when using Spearman's correlation end up with many tied ranks. It may therefore be more suitable to use Kendall's rank correlation which has the same properties as Spearman's correlation such as describing monotonic relationships, but Kendall's correlation also accounts for tied ranks. A pair of observations (x_i, y_i) and (x_j, y_j) where $i < j$ are called concordant if $x_i > x_j$ and $y_i > y_j$, or if $x_i < x_j$ and $y_i < y_j$. If $x_i = x_j$ or $y_i = y_j$ the pair is called tied. Disconcordant pairs are pairs that are not concordant or tied. Kendall's correlation coefficient is then defined as

$$\tau_{x,y} = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$$

where

-n_c is the number of concordant pairs

-n_d is the number of discordant pairs

-$n_0 = n(n-1)/2$, *i.e.*, the number of ways to choose two items from n items.

-$n_1 = \sum_i t_i(t_i - 1)/2$, where t_i is the number of tied values in the ith group of ties for x

- $n_2 = \sum_j u_j(u_j - 1)/2$, where u_j is the number of tied values in the jth group of ties in y.

4. **Elastic net**: elastic net is a shrinkage method for regression which combines l_1 and l_2 regularization (Zou and Hastie, 2005). l_1 regularization (often called Lasso) has the property of putting unimportant coefficients to zero, while l_2 regularization (often called ridge) shrinks correlated features equally. The hope is then that elastic net regularization will be able to shrink unimportant correlated features equally to zero. We can then use the non-zero features as input in our model.

Note that the number k is a hyperparameter that is tuned through the *feature_percentile* parameter (Table 3.1). We test the different feature selection methods using XGBoost and CatBoost.

4 Results

4.1 Comparison of Feature Selection Methods

The 10-fold cross-validation from the two-step procedure with bodymass as response indicates that the choice of feature selection method impacts the performance of the model greatly (Figure 4.1, left). Both XGBoost and CatBoost preform best when using Pearson's correlation coefficient as a selection method for the pre-selected SNPs, but they differ when it comes to which selection method they perform worst with. CatBoost with Spearman correlation has the lowest correlation distribution of all the selection methods considered here. XGBoost, however, handles the Spearman method better as it gives correlation scores comparable with the Pearson method. CatBoost also perform well with the Elasticnet method, while the Elasticnet method along with the Kendall method yield the lowest scores for XGBoost. CatBoost handles the Kendall method well as it is giving correlations on level with the Pearson method while having a low feature percentile.

Except for the Pearson method, it seems that XGBoost prefers to use more of the SNPs than CatBoost, as it overall has a higher median for feature percentile (Figure 4.1, right). However, both distributions of the feature percentile are quite wide, creating uncertainty about which feature percentile is the optimal one. We also see that there are many SNPs with little contribution when looking at the feature percentile of the elastic net method, where both XGBoost and CatBoost have feature percentiles near zero (Figure 4.1, right).

Note that the comparison of the different feature selection methods are only done with bodymass as response using the two-step approach. Ideally one would have compared the feature selection methods for tarsus length and for the one-step procedure as well, however, due to time constraints this was not feasible.

4.2 Results From Hyperparameter Tuning

For XGBoost with Pearson correlation as feature selection method and using the two-step procedure the tuned choice of hyperparameters differ depending on what target variable is used (Table 4.1). For the hyperparameters eta, subsample, colsample_bytree and feature_percentile we obtain similar values regardless of whether bodymass or tarsus length are modelled. When using bodymass as target variable, the alpha value is high while the lambda value is low, while when using tarsus as target variable both alpha and lambda has equal sizes. Both the alpha and the lambda parameter are regularizer's of the size of the weight in each node. Thus, when looking at the mean of lambda and alpha, we in total end up with approximate equal regularization of the weights. When looking at the parameters that regularize tree size, namely max_depth, min_child_weight and gamma, we have a similar effect as with the weight-regularizers. The model for bodymass require higher loss reduction (gamma) in order to split a node than the model for tarsus, while the tarsus model require a higher sum of weights in order to split the node (min_child_weight)

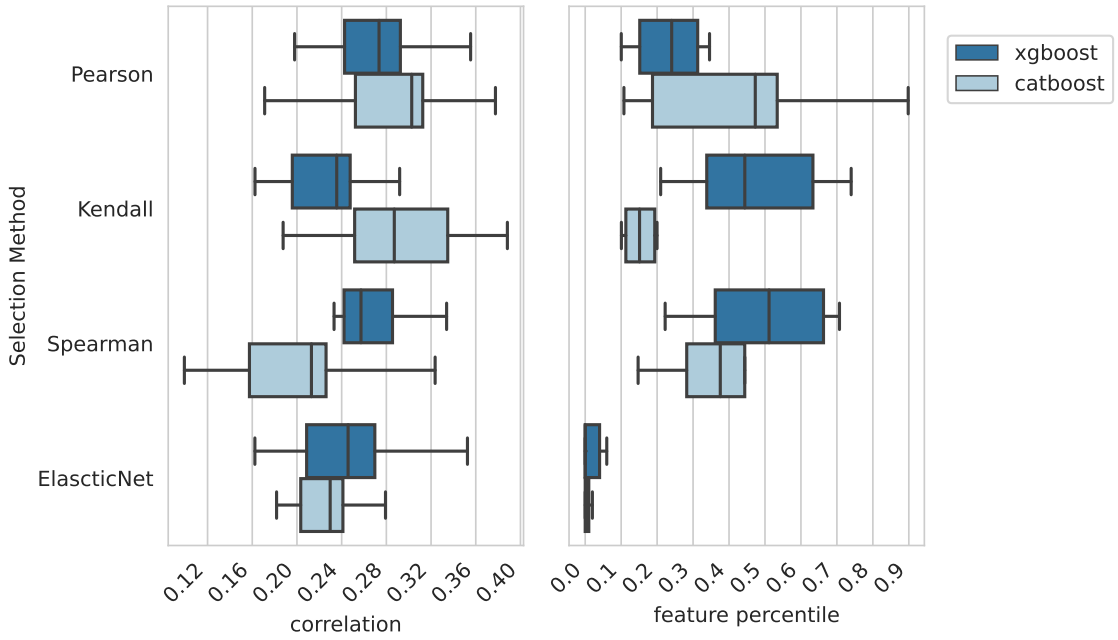

Figure 4.1: Comparison of feature selection methods. The y-axis shows the chosen feature selection method. The x-axis on the left shows correlation with the response. The x-axis on the right shows fraction of total number of SNPs used in model training

than the model for bodymass.

The colsample_bytree parameter control the number of features used when building each tree, while the feature_percentile parameter control the number of features that are used in total of the model. Using the values in Table 4.1 we get that the total number of SNPs available when building each tree is $182\ 854 \times 0.144 \times 0.7 \approx 18\ 431$ for the XGBoost bodymass model and $182\ 854 \times 0.22 \times 0.9 \approx 36\ 205$ for the XGBoost tarsus model. The tarsus model thus has about twice as many of the SNPs available when building each tree than what the bodymass model has. The max_depth parameter, however, restricts the number of SNPs that end up in the three and since the splits are binary, the maximal number of SNPs a tree can use is $2^{\text{max\_depth}}$. Small differences in max_depth therefore have big impact on the number of SNPs used. For each tree of the bodymass model, out of the 18 431 available SNPs only $2^5 = 32$ are at most used in each tree. If we now assume that each SNP only can be used once across all trees, we get that the bodymass model uses at most $2^{\text{max\_depth}} \times \text{n\_estimators} = 4896$ unique SNPs, while the tarsus length model uses at most 2416 unique SNPs.

Note that Table 4.1 only displays the chosen values used in one fold out of ten, the hyperparameters will differ a little from fold to fold. Therefore, one should not treat the hyperparameters displayed in Table 4.1 as an absolute truth, but rather look at the relative differences between the values chosen for bodymass and tarsus. Ideally the distribution of chosen hyperparameters from all folds should have been displayed, but due to an inconsistency in the experimental set up only the best performing model from each fold was saved, and due to constraints on time we have not been able to rerun the models. Similar tables as Table 4.1 for CatBoost, LightGBM and GBM are found in Section B.

Table 4.1: The tuned hyperparameters corresponding to the CV-fold with the highest correlation score for XGBoost using two-Step Method with Pearson correlation as feature selection method. The *name* column shows the name of the tuned Hyperparameter, the *description* column contains a short description of each hyperparameter that is tuned, and the *Tuned Value* column displays the best found hyperparameter for the best performing XGBoost model (out of 10) when using bodymass as target phenotype, and when using tarsus as target phenotype.

Name	**Description**	**Tuned Value**	
		Bodymass	**Tarsus**
max_depth	Maximum depth of a tree	5	4
alpha	L1 regularization parameter	6.76	2.21
lambda	L2 regularization parameter	0.0024	2.44
min_child_weight	Minimum sum of instance weight needed in a child	0.97	3.01
eta	The learning rate	0.044	0.065
subsample	The fraction of training observations used for each tree	0.75	0.65
n_estimators	The number of trees used	153	151
colsample_bytree	The fraction of features used for each tree	0.7	0.9
gamma	The threshold in loss reduction (Equation 2.14) required to split a node	0.81	0.001
feature_percentile	The fraction of preselected SNPs to be used in training	0.144	0.22

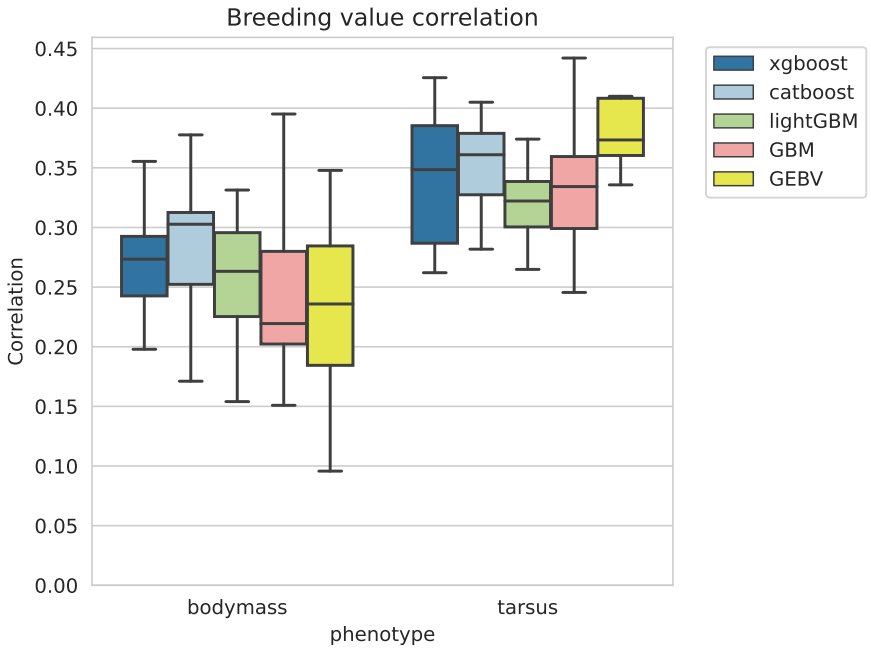

Figure 4.2: Correlation between the predicted breeding value and the pseduo-response y_i^* using the two-step procedure (see Section 3.2.2). All models (except the genomic animal model) are fitted using Pearson correlation as feature selection method. (left) correlations between predictions and target when phenotype is bodymass. (right) correlations between predictions and target when phenotype is tarsus.

4.3 Comparison of Model Performance

Using Pearson correlation for the pre-selection of SNPs we compare the performance of the different boosting models to each other and to the baseline genomic animal model. The Pearson correlation between predicted value of the response and actual value of the response is used as measure of model performance.

4.3.1 Two-Step Procedure

The 10-fold cross-validation from the two-step procedure with bodymass as response show that XGBoost, CatBoost and LightGBM are able to give higher correlations than the GEBVs from the fitted animal model (Figure 4.2, left). Ordinary gradient boosting (GBM), however, has the worst performance when using bodymass as response.

When using tarsus as response, all models are achieving noticeable higher correlations than when modelling bodymass (Figure 4.2, right). This is not surprising as tarsus is a trait with higher heritability compared to bodymass (Jensen, Steinsland et al., 2008). However, with tarsus length as target variable, the boosting models are not able to outperform the GEBVs from the animal model. Only XGBoost and CatBoost are achieving comparable correlations to the GEBVs from the genomic animal model. LightGBM and GBM yields lower model performance than the genomic animal model in terms of correlations when comparing the medians (Figure 4.2, right).

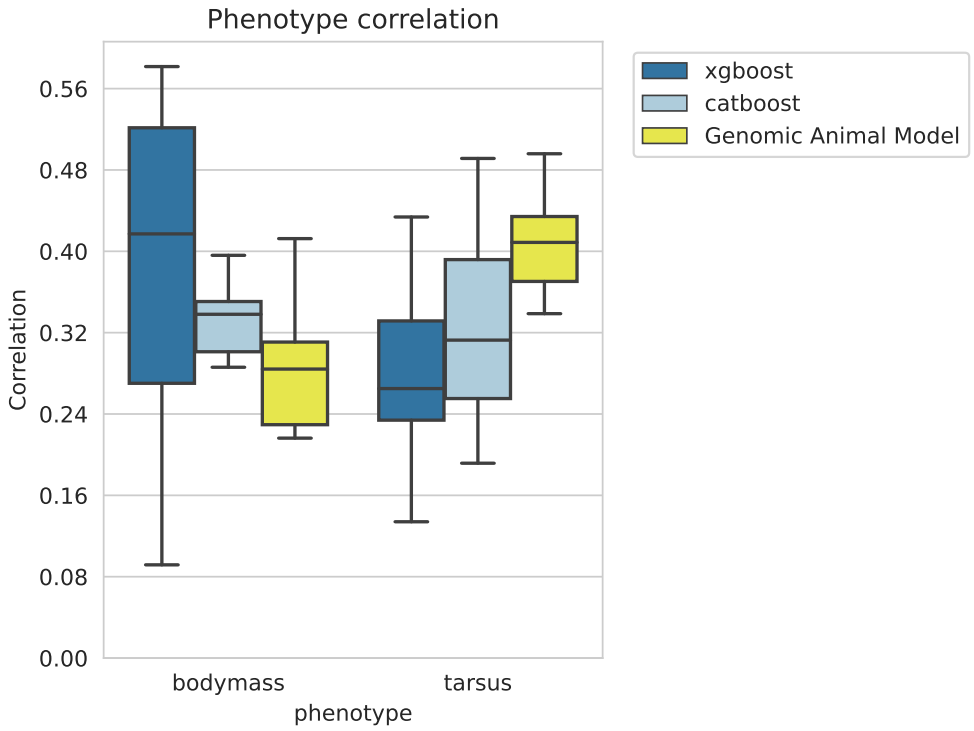

Figure 4.3: Correlation between the predicted phenotype and actual phenotype using the one-step procedure (see Section 3.2.3). All models (except the genomic animal model) are fitted using Pearson correlation as feature selection method. (left) correlation between predicted bodymass value and actual bodymass value. (right) correlation between predicted tarsus length and actual tarsus length

4.3.2 One-Step Procedure

For the one-step procedure, only XGBoost and CatBoost was fitted as they had the most promising results in the two-step procedure. The 10-fold cross validation from the one-step procedure with bodymass as response show again that XGBoost and CatBoost are able to outperform the genomic animal model (Figure 4.3, left). XGBoost have however large uncertainty, but shows potential to make accurate bodymass predictions. More surprisingly is the results when using tarsus as target, not only are the boosting models performing worse than the genomic animal model, they also have no improvement when comparing them to their bodymass-counterpart (Figure 4.3, right). Compared to the accuracies in the two-step procedure, we see that the boosting models for bodymass have an increase in performance when using the one-step procedure (Figure 4.3, left and Figure 4.2, left). The tarsus length boosting models, however, show a decrease in model performance when using the one-step procedure compared to the two-step procedure (Figure 4.3, right and Figure 4.2, right).

5 Discussion and Conclusions

In this project we have investigated different versions of boosted regression trees for the purpose of genomic prediction. We used two different approaches to model the phenotype of interest as a function genetic material. The first approach was a two-step procedure where genetic effects on the phenotype was isolated from the environmental effects, and the goal was to predict the genetic effects on the phenotype by using SNPs. The second approach aimed at predicting the phenotype directly by using a one-step procedure, where both the environmental effects and the genetic effects was included in the model by using SNPs and environmental data as predictors. For the two-step procedure we used the boosting methods XGBoost, CatBoost, LightGBM and GBM, while the one-step procedure used XGBoost and CatBoost. All models were compared to a Bayesian genomic animal model as benchmark. For both procedures, hyperparameter tuning was performed using Bayesian optimization and different methods for selection of SNPs prior to training was explored.

The results from the exploration of the feature selection method (Figure 4.1) show surprisingly that the Pearson correlation method performed the best, which is contradicting our hypothesis that the other selection methods should perform better due to a potential non-linear nature of correlation between the phenotype and the SNPs. As Pearson's correlation was used as a measure of the model quality, it was also used as an assessment of the different selection methods, thus it intuitively makes sense that the selection method using Pearson correlation perform best. Had another correlation coefficient been used, a different selection method may have been the best performing.

The differences between the performances of the selection methods are relatively small indicating that the choice of selection method does not have a big impact on model performance. This may partially be due to the boosting algorithm's inbuilt feature-wise subsampling where the methods themselves perform (although mostly random) feature selection. However, it is more likely due to the nature of decision trees, where the trees actively seek the features that give them the best split. By limiting how deep the trees are allowed to grow we actively limit the number of features that can be used, and thus indirectly perform feature selection. As calculated from Table 4.1, the best performing XGBoost model for tarsus length using the two-step procedure is using at most 2416 unique SNPs, while the model for bodymass can at most use 4896 unique SNPs. The big difference in effective SNP size can be interpreted as an illustration of the difference in genetic complexity between bodymass and tarsus length as phenotypes. Bodymass is a highly volatile trait where it is likely that many factors play a role, while tarsus length is a trait that remains more or less constant once the individual in question has reached adulthood. One should, however, be careful with drawing conclusions from hyperparameters as they only explain the architecture of the model and not necessarily the data. The models in question are, however, performing on par with the widely used animal model, and similar explanations has been proposed to explain differences between traits in genomic prediction accuracy (Ashraf et al., 2022), making the conclusions we draw more reliable.

The results from the two-step procedure show that the boosting algorithms XGBoost and

CatBoost are able to perform on level with the GEBVs from the animal model when modelling the tarsus length, and when modelling bodymass XGBoost and CatBoost are able to outperform the GEBVs. LightGBM is able to give performance comparable to the GEBVs from the animal model when modelling bodymass, but yield worse performance when using tarsus length as response. GBM perform worse than the GBLUPs for both bodymass and tarsus length. The reason for the boosting models to have varying results relative to the animal model when looking at different traits may be due to the complexity of the traits. As discussed, tarsus length seems to be less polygenetic compared to bodymass. Therefore, it might be that tarsus length is a trait explained more by additive genetic effects while bodymass is also heavily reliant on epistatic effects. This assumption is supported by the difference between the max_depth parameter which also control the degree of interactions between the features. The deeper a tree is, the more interactions are modelled. In the case of additive genetic effects, a linear model, which models the SNPs as independent additive effects, will be the more appropriate model for tarsus length. Whereas in the case of epistasis a model which allows for complex interactions will be more suitable for bodymass.

The one-step procedure gave a boost in the performance of the XGBoost and CatBoost models for bodymass when compared to the one-step procedure. Especially XGBoost had a big improvement. The one-step procedure modelled both environmental variables and SNPs directly and together, which allows for modelling of the interaction between genes and environment. The increase in model performance may therefore be due to the inclusion of gene-environment interaction, and indicates that bodymass is a trait dependent on gene-environment interactions. The one-step models for tarsus, however, performed both worse than the bodymass models, but also worse than the two-step models for tarsus. Tarsus length have higher heritability than then bodymass (Jensen, Steinsland et al., 2008), meaning that the variability in tarsus length is more explained by variation due to genetic effects than for bodymass. It could thus be that the boosting models overestimate the environmental effects in the one-step procedure for tarsus length, leading to worse prediction performance.

More investigation of the feature importance in the boosting models is needed. Feature importance can be done using XAI tools such as SHAP, (Molnar, 2022), allowing us to find out which SNPs and environmental variables are contributing the most to the model output. An interesting property of the most important SNPs would be the location of the SNP in the genetic material as this would be a strong indicator of what kinds of genetic effects are influencing the phenotype.

In Section 3.2.3 we argued that it were important to do the cross-validation based on the ringnr-indexes to avoid problems with repeated individuals. It is worth noting that the one-step approach is still vulnerable to potential clusters in other variables. Clusters are gatherings of groups in the data where the response variable varies more between clusters than within. This sounds initially as a good thing, since trees are looking for ways to cluster together the target variables based on the predictors. However, clusters in the data can create a problem when the clusters are due to a categorical variable. If one category have far more observations than the other categories we end up with class-inbalance. Class-inbalance creates a problem when splitting the data into train, test and validation sets as we may risk ending up with a training set containing a non-representative picture of the number of observations per category. The non-representative picture can potentially make the model over- or underestimate the effect of the categorical variable.

As a concrete example of a potential cluster and the class-inbalance that may arise, con-

sider that the house sparrows live at different islands which have different environmental conditions. As there are generally more native sparrows than immigrants on each island, the sparrows are also more likely to mate with native residents than immigrants. Therefore, it is more likely that two sparrows from the same island will have similar genetic material than two sparrows from different islands. As we know from quantitative genetics, the phenotype is a function of environmental and genetic factors (Conner and Hartl, 2004b). Thus, if the environmental and genetic factors differ more between islands than within, it is not far-fetched to believe that the phenotype will differ more between island populations than within island populations. The island location is hence an example of a potential cluster that can appear in the data. The potential danger is that the islands are not evenly populated, resulting in class-inbalance where some islands having far more observations than others. We may risk ending up with a train set which does not contain observations from the sparsely populated islands, making the model prone to underestimate the effect of island location. We may also end up with the opposite, that the train set by chance contains equal number of observations from the sparsely populated islands as the bigger islands, which would make the model overestimate the effect of island location.

There are some limitations to the experimental setup used in this project when it comes to the hyperparameter tuning procedure. The hyperparameter values one end up with in each fold is dependent on the random starting point and the number of trials used in the Bayesian optimization. As hyperparameter tuning is a time-consuming activity we chose to use 30 optimization trials which may be a too small number, as a reference Prokhorenkova et al. (2017) use 50 optimization trials when comparing XGBoost, CatBoost and LightGBM. More trials would be more time-consuming, but allows for further exploration of promising hyperparameter values. Prokhorenkova et al. (2017) also use a 5-fold cross-validation in the hyperparameter tuning to get greater power in their assessment of the selected hyperparameters.

The starting values of the optimization also impacts which hyperparameters one end up with in the end, and as we use a fixed seed to ensure comparability between models, all folds use the same starting point. Therefore, to allow for more exploitation of the different combinations of the hyperparameters, a different experimental set up may be preferable. One could first do cross-validation where one perform hyperparameter tuning using different (but predetermined) seeds in each fold. Then we choose the best set of hyperparameters from the cross-validation before running a new cross-validation, where in each fold the model is trained using the same set of hyperparameters. Such an approach would allow for more exposure of different combinations of starting points, but one must be careful when splitting the data to avoid testing the model on data that have been used to select hyperparameters. Another potential improvement of the hyperparameter tuning set-up is the assessment criterion. We used the mean squared error (mean of (2.6)) to assess the quality of the hyperparameters, however the model's performance was in the end assessed using the Pearson correlation (3.1). To maximize the model performance it may therefore have been more suitable to use the Pearson correlation as an assessment of the hyperparameters as well.

To improve the accuracy of traits like tarsus length, where we concluded that an additive model is more appropriate, one could either set stronger enforcements on the tree depth to limit interactions and then increase the number of boosting iterations to allow for more additivity. However, a smarter approach may be to replace the trees with linear regression as the weak learners in the boosting. Linear regression boosting could be seen as an extension of the two-step approach where we first use a (mixed) linear regression and then instead of using regression trees we now use linear regression models. Using linear

regression in boosting would give a model which is additive in nature, while maintaining the properties of boosting where one iteratively improves the accuracy. To address the issue of tarsus length having poorer performance when including the environmental effects one could consider other modelling set-ups. For instance, one could create a meta-ensemble where the genetic variables and environmental effects on the phenotype are modelled separately before they are merged into one prediction.

In this project we have found that boosting algorithms can be a realistic alternative to the animal model in terms of genomic prediction accuracy. Further work based on this project would be to explore more ways to set up the hyperparameter search, such as include nested cross-validation and increase the number of optimization trials. We would be sure to save all results from the hyperparameter tuning during the cross-validation to get more insight in the variability of the final hyperparameters, which would give more power in our analysis. Feature importance of both SNPs and environmental variables is also important to investigate as this would allow for more biological explanations of the behavior of the model's prediction of the traits. To address issues of cluster's in the data we would explore methods such as stratified cross-validation which combats class-inbalance (Prusty et al., 2022). Lastly we would investigate other boosting methods such as boosting of linear regression models, and explore other ways to model environmental and genetic effects such as creating a meta-ensemble.

Bibliography

Brooks, R. L. (1941). 'On colouring the nodes of a network'. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 37.2, pp. 194–197. DOI: 10.1017/S030500410002168X.

Curry, Haskell B. (1944). 'The method of steepest descent for non-linear minimization problems'. In: *Quarterly of Applied Mathematics* 2.258-261.

Kendall, M. G. (Nov. 1945). 'The Treatment Of Ties In Ranking Problems'. In: *Biometrika* 33.3, pp. 239–251. ISSN: 0006-3444. DOI: 10.1093/biomet/33.3.239. eprint: https://academic.oup.com/biomet/article-pdf/33/3/239/573257/33-3-239.pdf. URL: https://doi.org/10.1093/biomet/33.3.239.

Henderson, C (1950). 'Estimation of Genetic Parameters'. In: *The Annals of Mathematical Statistics* 21.309-310.

Henderson, C. R. (June 1975). 'Best Linear Unbiased Estimation and Prediction under a Selection Model'. In: *Biometrics* 31.2, p. 423. ISSN: 0006341X. DOI: 10.2307/2529430.

Breiman, Leo et al. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Henderson, C.R. (1984). *Applications of Linear Models in Animal Breeding*. University of Guelph.

Falconer, Douglas Scott (1996). *Introduction to quantitative genetics*. Pearson Education India.

Lynch, Michael and Bruce Walsh (1998). *Genetics and analysis of quantitative traits*. Vol. 1. Sinauer Sunderland, MA.

Mason, Llew et al. (1999). 'Boosting algorithms as gradient descent'. In: *Advances in neural information processing systems* 12.

Friedman, Jerome H. (Oct. 2001). 'Greedy function approximation: A gradient boosting machine.' In: *The Annals of Statistics* 29.5. ISSN: 0090-5364. DOI: 10.1214/aos/1013203451.

Meuwissen, T H E, B J Hayes and M E Goddard (Apr. 2001). 'Prediction of Total Genetic Value Using Genome-Wide Dense Marker Maps'. In: *Genetics* 157.4, pp. 1819–1829. ISSN: 1943-2631. DOI: 10.1093/genetics/157.4.1819.

Casella, George and Roger L. Berger (2002). 'Bayes Estimators'. In: *Statistical Inference*. 2nd ed. Pacific Grove, CA: Duxbury. Chap. 7, pp. 324–326.

Friedman, Jerome H. (2002). 'Stochastic gradient boosting'. In: *Computational Statistics and Data Analysis* 38.4. Nonlinear Methods and Data Mining, pp. 367–378. ISSN: 0167-9473. DOI: https://doi.org/10.1016/S0167-9473(01)00065-2. URL: https://www.sciencedirect.com/science/article/pii/S0167947301000652.

Kruuk, Loeske E. B. et al. (Aug. 2002). 'Antler size in red deer: heritability and selection but no evolution'. In: *Evolution* 56.8, pp. 1683–1695. ISSN: 0014-3820. DOI: 10.1111/j.0014-3820.2002.tb01480.x.

Conner, Jefferey K. and Daniel L. Hartl (2004a). 'Population Genetics I: Genetic Variation, random and nonrandom mating'. In: *A Primer of Ecological Genetics*. 1st ed. Sunderland: Sinauer Associates. Chap. 4, pp. 9–42.

— (2004b). 'Quantitative Genetics I'. In: *A Primer of Ecological Genetics*. 1st ed. Sunderland: Sinauer Associates. Chap. 4, pp. 99–112.

Connor, Jefferey K. and Daniel L. Hartl (2004). 'Introduction'. In: *A Primer of Ecological Genetics*. 1st ed. Sunderland, Massachusetts: Sinauer Associates. Chap. 1, pp. 1–2.

Jensen, Henrik, Bernt-Erik Sæther et al. (July 2004). 'Lifetime reproductive success in relation to morphology in the house sparrow '. In: *Journal of Animal Ecology* 73.4, pp. 599–611. ISSN: 0021-8790. DOI: 10.1111/j.0021-8790.2004.00837.x.

Kruuk, Loeske E. B. (June 2004). 'Estimating genetic parameters in natural populations using the 'animal model''. In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 359.1446, pp. 873–890. ISSN: 0962-8436. DOI: 10.1098/rstb.2003.1437.

Zou, Hui and Trevor Hastie (Mar. 2005). 'Regularization and Variable Selection Via the Elastic Net'. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.2, pp. 301–320. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2005.00503.x. eprint: https://academic.oup.com/jrsssb/article-pdf/67/2/301/49795094/jrsssb_67_2_301.pdf. URL: https://doi.org/10.1111/j.1467-9868.2005.00503.x.

Gamerman, Danim and Hedibert F. Lopes (2006). 'Bayesian Inference'. In: *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman and Hall/CRC.

Walker, Francis O (Jan. 2007). 'Huntington's disease'. In: *The Lancet* 369.9557, pp. 218–228. ISSN: 01406736. DOI: 10.1016/S0140-6736(07)60111-1.

Jensen, Henrik, Ingelin Steinsland et al. (June 2008). 'Evolutionary dynamics of a sexual ornament in the house sparrow (passer domesticus): the role of indirect selection within and between sexes'. In: *Evolution* 62.6, pp. 1275–1293. ISSN: 0014-3820. DOI: 10.1111/j.1558-5646.2008.00395.x.

VanRaden, P.M. (Nov. 2008). 'Efficient Methods to Compute Genomic Predictions'. In: *Journal of Dairy Science* 91.11, pp. 4414–4423. ISSN: 00220302. DOI: 10.3168/jds.2007-0980.

Hastie, Trevor, Robert Tibshirani and Jerome Friedman (2009). *The Elements of Statistical Learning*. New York, NY: Springer New York. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7.

Manolio, Teri A et al. (Oct. 2009). 'Finding the missing heritability of complex diseases.' In: *Nature* 461.7265, pp. 747–53. ISSN: 1476-4687. DOI: 10.1038/nature08494.

Rue, Håvard, Sara Martino and Nicolas Chopin (Apr. 2009). 'Approximate Bayesian Inference for Latent Gaussian models by using Integrated Nested Laplace Approximations'. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 71.2, pp. 319–392. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2008.00700.x.

Chevin, Luis-Miguel, Russell Lande and Georgina M. Mace (Apr. 2010). 'Adaptation, Plasticity, and Extinction in a Changing Environment: Towards a Predictive Theory'. In: *PLoS Biology* 8.4, e1000357. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.1000357.

Steinsland, Ingelin and Henrik Jensen (2010). 'Utilizing Gaussian Markov Random Field Properties of Bayesian Animal Models'. In: *Biometrics* 66.3, pp. 763–771. DOI: https://doi.org/10.1111/j.1541-0420.2009.01336.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1541-0420.2009.01336.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1541-0420.2009.01336.x.

Wilson, Alastair J. et al. (Jan. 2010). 'An ecologist's guide to the animal model'. In: *Journal of Animal Ecology* 79.1, pp. 13–26. ISSN: 0021-8790. DOI: 10.1111/j.1365-2656.2009.01639.x.

Baccouche, Moez et al. (2011). 'Sequential Deep Learning for Human Action Recognition'. In: pp. 29–39. DOI: 10.1007/978-3-642-25446-8_4.

Bergstra, James, Rémi Bardenet et al. (2011). 'Algorithms for Hyper-Parameter Optimization'. In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Bergstra, James and Yoshua Bengio (2012). 'Random Search for Hyper-Parameter Optimization'. In: *Journal of Machine Learning Research* 13.10, pp. 281–305. URL: http://jmlr.org/papers/v13/bergstra12a.html.

Pedregosa, Fabian et al. (Jan. 2012). 'Scikit-learn: Machine Learning in Python'. In.

Geof H. Givens and Jennifer A. Hoeting (2013). 'Markov Chain Monte Carlo'. In: *Computational Statistics*. 2nd ed. John Wiley & Sons. Chap. 7.

Holand, Anna Marie et al. (Aug. 2013). 'Animal Models and Integrated Nested Laplace Approximations'. In: *G3 Genes—Genomes—Genetics* 3.8, pp. 1241–1251. ISSN: 2160-1836. DOI: 10.1534/g3.113.006700. eprint: https://academic.oup.com/g3journal/article-pdf/3/8/1241/37068182/g3journal1241.pdf. URL: https://doi.org/10.1534/g3.113.006700.

Charmantier, Anne, Dany Garant and Loeske EB Kruuk (2014). *Quantitative genetics in the wild*. OUP Oxford.

Chen, Tianqi and Carlos Guestrin (Aug. 2016). 'XGBoost'. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://dl.acm.org/doi/10.1145/2939672.2939785.

Kearnes, Steven et al. (Aug. 2016). 'Molecular graph convolutions: moving beyond fingerprints'. In: *Journal of Computer-Aided Molecular Design* 30.8, pp. 595–608. ISSN: 0920-654X. DOI: 10.1007/s10822-016-9938-8.

Meuwissen, Theo, Ben Hayes and Mike Goddard (Jan. 2016). 'Genomic selection: A paradigm shift in animal breeding'. In: *Animal Frontiers* 6.1, pp. 6–14. ISSN: 2160-6056. DOI: 10.2527/af.2016-0002.

Albrecht, Tim et al. (Oct. 2017). 'Deep learning for single-molecule science'. In: *Nanotechnology* 28.42, p. 423001. ISSN: 0957-4484. DOI: 10.1088/1361-6528/aa8334.

Ke, Guolin et al. (2017). 'LightGBM: A Highly Efficient Gradient Boosting Decision Tree'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., pp. 3149–3157.

Prokhorenkova, Liudmila et al. (June 2017). 'CatBoost: unbiased boosting with categorical features'. In: URL: http://arxiv.org/abs/1706.09516.

Frazier, Peter I. (July 2018). 'A Tutorial on Bayesian Optimization'. In.

Li, Bo et al. (2018). 'Genomic Prediction of Breeding Values Using a Subset of SNPs Identified by Three Machine Learning Methods'. In: *Frontiers in Genetics* 9. ISSN: 1664-8021. DOI: 10.3389/fgene.2018.00237. URL: https://www.frontiersin.org/articles/10.3389/fgene.2018.00237.

Lundregan, Sarah L. et al. (Sept. 2018). 'Inferences of genetic architecture of bill morphology in house sparrow using a high-density SNP array point to a polygenic basis'. In: *Molecular Ecology* 27.17, pp. 3498–3514. ISSN: 0962-1083. DOI: 10.1111/mec.14811.

Martino, Sara and Andrea Riebler (2019). *Integrated Nested Laplace Approximations (INLA)*. arXiv: 1907.01248 [stat.CO].

Muff, Stefanie et al. (Dec. 2019). 'Animal models with group-specific additive genetic variances: extending genetic group models'. In: *Genetics Selection Evolution* 51.1, p. 7. ISSN: 1297-9686. DOI: 10.1186/s12711-019-0449-7.

Ge, M. et al. (Aug. 2020). 'Deep learning analysis on microscopic imaging in materials science'. In: *Materials Today Nano* 11, p. 100087. ISSN: 25888420. DOI: 10.1016/j.mtnano.2020.100087.

Johnsen, Pål V. et al. (Dec. 2021). 'A new method for exploring gene–gene and gene–environment interactions in GWAS with tree ensemble methods and SHAP values'. In: *BMC Bioinformatics* 22.1, p. 230. ISSN: 1471-2105. DOI: 10.1186/s12859-021-04041-7.

Montesinos-López, Osval Antonio et al. (Jan. 2021). 'A review of deep learning applications for genomic selection'. In: *BMC Genomics* 22.1, p. 19. ISSN: 1471-2164. DOI: 10.1186/s12864-020-07319-x.

Ashraf, Bilal et al. (Dec. 2022). 'Genomic prediction in the wild: A case study in Soay sheep'. In: *Molecular Ecology* 31.24, pp. 6541–6555. ISSN: 0962-1083. DOI: 10.1111/mec.16262.

Gill, Mitchell et al. (Apr. 2022). 'Machine learning models outperform deep learning models, provide interpretation and facilitate feature selection for soybean trait prediction'. In: *BMC Plant Biology* 22.1, p. 180. ISSN: 1471-2229. DOI: 10.1186/s12870-022-03559-z.

Molnar, Christoph (2022). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. URL: https://christophm.github.io/interpretable-ml-book.

Nazzicari, Nelson and Filippo Biscarini (Nov. 2022). 'Stacked kinship CNN vs. GBLUP for genomic predictions of additive and complex continuous phenotypes'. In: *Scientific Reports* 12.1, p. 19889. ISSN: 2045-2322. DOI: 10.1038/s41598-022-24405-0.

Prusty, Sashikanta, Srikanta Patnaik and Sujit Kumar Dash (2022). 'SKCV: Stratified K-fold cross-validation on ML classifiers for predicting cervical cancer'. In: *Frontiers in Nanotechnology* 4. ISSN: 2673-3013. DOI: 10.3389/fnano.2022.972421. URL: https://www.frontiersin.org/articles/10.3389/fnano.2022.972421.

Virolainen, Samuel J. et al. (Dec. 2022). 'Gene–environment interactions and their impact on human health'. In: *Genes & Immunity* 24.1, pp. 1–11. ISSN: 1476-5470. DOI: 10.1038/s41435-022-00192-6.

Chafai, Narjice et al. (2023). 'A review of machine learning models applied to genomic prediction in animal breeding'. In: *Frontiers in Genetics* 14. ISSN: 1664-8021. DOI: 10.3389/fgene.2023.1150596. URL: https://www.frontiersin.org/articles/10.3389/fgene.2023.1150596.

Munar-Delgado, Gabriel, Yimen G. Araya-Ajoy and Pim Edelaar (May 2023). 'Estimation of additive genetic variance when there are gene–environment correlations: Pitfalls, solutions and unexplored questions'. In: *Methods in Ecology and Evolution* 14.5, pp. 1245–1258. ISSN: 2041-210X. DOI: 10.1111/2041-210X.14098.

Bergstra, James, Dan Yamins and David D Cox (n.d.). *Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms*. Tech. rep. 1.

Appendix

A Hyperparameter Distributions for CatBoost, LightGBM and GBM

Table 1: Hyperparameter Distributions for CatBoost

Name	**Description**	**Distribution**
learning_rate	The learning rate in each boosting iteration	Log-uniform at $[e^{-7}, 1]$
random_strength	The amount of randomness to add when evaluating possible split points.	Uniform at $[0, 20]$
l2_leaf_reg	L2 regularization parameter on weights	Log-uniform at $[1, 10]$
bagging temperature	Controls the intensity of the bagging, *i.e.*, subsampling of observations	Uniform at $[0, 1]$
leaf estimation iterations	The optimal weight value (2.12) is essentially a newton optimization, thus multiple iterations are possible to perform, which is what this parameter controll.	Random integers from $[1, 10]$
feature_percentile	Fraction of SNPs used in booster	Uniform at $[0.1, 0.9]$

Table 2: Hyperparameter Distributions for LightGBM

Name	**Description**	**Distribution**
max_depth	Maximum depth of a tree	Random integers from $[15, 10000]$
reg_alpha	L1 regularization parameter on weights	Log-uniform at $[e^{-8}, e^{2}]$
reg_lambda	L2 regularization parameter on weights	Log-uniform at $[e^{-8}, e^{2}]$
num_leaves	Maximum number of leaves in a tree	Random integers from $[10, 10000]$
learning_rate	The learning rate	Log-uniform at $[e^{-7}, 1]$
subsample	The fraction of training data used for each tree	Uniform at $[0.5, 1]$
n_estimators	The number of trees used	Random integers from $[20, 205]$
colsample_bytree	The fraction of columns used for each tree	Uniform at $[0.5, 1]$
min_data_in_leaf	The minimum number of observations belonging to a leaf node during training	random integers from $[1, 5000]$
min_sum_hessian in_leaf	The minimum sum of the hessian required to split a leaf node	Log-uniform at $[-15, 5]$
feature_percentile	Fraction of SNPs used in booster	Uniform at $[0.1, 0.9]$

Table 3: Hyperparameter Distributions for GBM

Name	**Description**	**Distribution**
max_depth	Maximum depth of a tree	Random integers from $[5, 30]$
learning_rate	The learning rate	Log-uniform at $[e^{-7}, 1]$
subsample	The fraction of training data used for each tree	Uniform at $[0.5, 1]$
n_estimators	The number of trees used	Random integers from $[20, 205]$
min_weight fraction_leaf	The minimum fraction of the the sum of toatl weights of a weight required to be a leaf node.	Uniform at $[0, 0.45]$
feature_percentile	Fraction of SNPs used in booster	Uniform at $[0.1, 0.9]$

B Tuned Hyperparameter Values for CatBoost and LightGBM

Table 4: The tuned hyperparameters corresponding to the CV-fold with the highest correlation score for LightGBM using two-Step Method with Pearson correlation as feature selection method.

Name	**Description**	**Tuned Value**	
		Bodymass	**Tarsus**
max_depth	Maximum depth of a tree	547	68
reg_alpha	L1 regularization parameter	0.0193	0.0023
reg_lambda	L2 regularization parameter	0.0064	0.896
num_leaves	Maximum number of leaves in a tree	339	909
learning_rate	The learning rate	0.058	0.0225
subsample	The fraction of training observations used for each tree	0.833	0.694
n_estimators	The number of trees used	80	89
colsample_bytree	The fraction of features used for each tree	0.51	0.565
min_data_in_leaf	The minimum number of observations belonging to a leaf node during training	49	62
min_sum_hessian in_leaf	The minimum sum of the hessian required to split a leaf node	4.468	0.033
feature_percentile	The fraction of preselected SNPs to be used in training	0.159	0.313

Table 5: The tuned hyperparameters corresponding to the CV-fold with the highest correlation score for CatBoost using two-Step Method with Pearson correlation as feature selection method.

Name	**Description**	**Tuned Value**	
		Bodymass	**Tarsus**
learning_rate	The learning rate in each boosting iteration	0.0584	0.1103
random_strength	The amount of randomness to add when evaluating possible split points	5.0	17.0
l2_leaf_reg	L2 regularization parameter on weights	428.25	6.31
bagging temperature	Controls the intensity of the bagging, *i.e.*, subsampling of observations	0.643	0.85
leaf estimation iterations	The optimal weight value (2.12) is essentially a newton optimization, thus multiple iterations are possible to perform, which is what this parameter control.	6	1
feature_percentile	The fraction of preselected SNPs to be used in training	0.183	0.284

Table 6: The tuned hyperparameters corresponding to the CV-fold with the highest correlation score for CatBoost using two-Step Method with Pearson correlation as feature selection method.

Name	**Description**	**Tuned Value**	
		Bodymass	**Tarsus**
max_depth	Maximum depth of a tree	11	15
learning_rate	The learning rate	0.117	0.140
subsample	The fraction of training data used for each tree	0.55	0.85
n_estimators	The number of trees used	175	202
min_weight fraction_leaf	The minimum fraction of the the sum of toatl weights of a weight required to be a leaf node.	0.30	0.13
feature_percentile	Fraction of SNPs used in booster	0.170	0.487