Universidad de la República
Facultad de Ingeniería

DN*AI*: Machine Learning for Genome Enabled Prediction of complex traits in agriculture.

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Juan Elenter, Guillermo Etchebarne, Ignacio Hounie

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Maria Inés Fariello ..... Universidad de la República
Federico Lecumberry ..... Universidad de la República

Tribunal

.....

Montevideo
Monday 31st May, 2021

DNAI: Machine Learning for Genome Enabled Prediction of complex traits in agriculture., Juan Elenter, Guillermo Etchebarne, Ignacio Hounie.

This thesis was wriitten in LaTeX using the iietesis (v1.1) class.
It contains 213 pages.
Compiled Monday 31st May, 2021.
http://iie.fing.edu.uy/

I find this case very confusing, and have not thoroughly checked the result.

R.A.Fisher

This page intentionally left blank.

Acknowledgements

This work was partially funded by project ANII FSDA 1-2018-1-154364.

The experiments presented in this work were carried out using ClusterUY (site: https://cluster.uy).

This page intentionally left blank.

To all the women that made invaluable early contributions to quantitative and computational genetics (Dung et al., 2019), and whose names should be in the references.

This page intentionally left blank.

Abstract

Genome enabled prediction of complex traits aims to predict a measurable characteristic of an organism using their genetic information. In the present work we address diverse traits and organisms including yeast growth, wheat yield, Jersey bull fertility and Holstein cattle milk yield.

We benchmark several popular Machine Learning models: bayesian and penalized linear regressions, kernel methods, and decision tree ensembles. Through exhaustive hyperparameter tuning we outperform state-of-the-art results in most datasets. We also compare two codification techniques for input data and perform ablation studies to assess robustness to genetic marker - i.e input features - elimination.

We then explore different Deep Learning architectures for this task. We propose and evaluate CNN architectures, showing that using residual connections improves perfomance but that in some cases Fully Connected Networks outperform CNNs. We link this to the fact that absolute positions are relevant in genomes, and thus, CNN's translational equivariance may not be an adequate inductive bias for tackling this problem.

In addition, we explore using PCA and TSNE for mapping input features to two-dimensional image-like feature maps used as inputs to 2D-CNN architectures. We assess the effectiveness of the aforementioned dimensionality reduction techniques when used to construct those mappings, and find that in some cases, using random mappings performs comparably. We also propose a method to construct these image-like feature maps based on an approximation to the Fermat distance.

Furthermore, we evaluate graph neural network architectures by formulating trait prediction as a node regression problem on a population graph, where each node represents an individual, and edges association between their genetic information. We evaluate the transferability of these graphical models and find that the extent to which they exploit neighbourhood information is limited. We also propose a model combining CNN and GNN architectures, which outperforms all other models in Holstein cattle milk yield prediction.

Lastly, we propose optimising Pearson correlation directly, which is commonly used to evaluate model performance, but MSE is usually minimised. Although this loss does not penalise learning an affine transformation of actual phenotypes, we show that this affine transformation can be estimated from train data, and leads to models with both lower MSE and higher predictive correlations.

This page intentionally left blank.

Foreword

Millions of years of evolution have been enabled by the most dense and durable (Heckel, Shomorony, Ramchandran, & David, 2017) information storage medium known to date: the DNA. From a cell in a toenail to the most complex neuron in the brain, from breathing to subjective wellbeing (Bartels, 2015), the information behind biological processes that take place on every single living organism throughout its lifetime is encoded in a single molecule.

Our understanding about it is still (strikingly) fresh. Although the first whole DNA sequence dates from 1965 it was not until 2001 that the human genome was completely mapped (Heather & Chain, 2016). We are just beginning to move on from a few known sites in the chain to a fuller picture of the genetic instructions that govern all life on earth.

As far fetched as it may seem, genome-enabled prediction already has a myriad of interesting applications ranging from medicine (Evans, Visscher, & Wray, 2009; Hindorff et al., 2009; Ritchie, 2012) to agriculture (Crossa et al., 2010; Qanbari et al., 2011; Weigel, VanRaden, Norman, & Grosu, 2017).

The availability of genetic information is very likely to continue growing at astounding rates, as costs keep falling (Akdemir & Isidro-Sánchez, 2019) and genotyping technologies keep improving. So will computing power, even in a post-Moore era (Leiserson et al., 2020). What remains uncertain is to what extent will science be able to exploit both trends in conjunction.

We share one belief: it is worth finding out.

This page intentionally left blank.

Objectives

We propose to explore modern machine learning techniques, with an emphasis in Deep Learning, in the task of Genetic Prediction of Complex traits. That is, simply put, predicting a measurable characteristic of an individual based on information about its genome. We do not focus on using external sources of information about the populations, environments, or domain knowledge about the underlying biological processes.

We focus on traits with agricultural value. This stems from the the fact that this thesis was proposed in the context of a research project concerning the prediction of traits related to the quality of bovine meat. Genome enabled Prediction in agriculture can lead not only to an increase in quality (Weigel et al., 2017), but also to an increase in resource efficiency which may be key to sustainability (Bohra, Chand Jha, Godwin, & Kumar Varshney, 2020).

To enable a richer analysis, we choose *four* datasets with different characteristics, out of which three belong to the agricultural domain.

Assessing which inductive biases may be beneficial for the task at hand is crucial for determining future research directions, especially in a field where Deep Learning is still incipient. In that sense, we explore two main types of Neural Network architectures: Convolutional Neural Networks and Graph Neural Networks.

We also set out to discuss metrics, loss functions, optimization algorithms and hyperparameter settings, which greatly affect *'performance'* for all models and are thus a core aspect of the problem.

This page intentionally left blank.

Document Overview

This document is structured as follows. We begin by introducing fundamental quantitative genetics concepts, with the aim of introducing the Genomic Prediction problem on Chapter 1. We then formulate this problem mathematically on Chapter 2. We also present performance metrics and loss functions which are part of the problem formulation as well as optimization algorithms which will be used to train several models.

Chapter 3 describes the particular datasets that were chosen to tackle this problem. As it will be shown later, this choice has a great impact on results. After that, we include a brief review of predictive models in Genomic literature in Chapter 4. We describe the most common methods, introducing domain-specific terms and references. These same models will then be used as baselines and evaluated on the already described datasets.

On chapter 6 we begin to explore Neural Networks and Deep learning, by presenting both one and two dimensional Convolutional Neural Networks (CNNs). This Chapter includes a literature review, a description of the proposed architectures, and the experiments that were carried out.

On Chapter 7 we formulate Genomic Prediction as a graph regression problem. We examine graph topology inference in that setting, and present Graph Neural Network (GNN) architectures and several experiments.

Lastly, Chapter 8 lays out conclusions and further work.

This page intentionally left blank.

Contents

Chapter 1

Genetics basics

In this chapter some basic genetics concepts are introduced on Section 1.1. Then Section 1.2 lays out quantitative genetic fundamentals, which set the groundwork for genetic prediction. Further details can be found on several great introductory texts such as Falconer and Mackay (1996); Griffiths, Wessler, Carroll, and Doebley (2015).

1.1 DNA 101

Until the late 19^{th} century, there was no reasonable explanation for the apparently innocuous observation: "children resemble their parents". The transmission of traits was an empirical fact often used in agriculture (Palladino, 1993; Theunissen, 2008), but the underlying mechanisms enabling the storage and transmission of biological information were unknown. In 1866, an Austrian monk called Gregor Mendel suggested the existence of discrete units of inheritance, and successfully explained the transmission of qualitative traits, such as color in pea plants. This triggered decades of research that would shed light on the biological information mystery, now known as genetics.

Genetic information is encoded in a double-stranded molecule called DNA (**D**eoxyribo**N**ucleic **A**cid). Each strand is composed of simpler units called nucleotides. In turn, each nucleotide contains one of four nitrogen bases: Adenine, Thymine, Guanine or Cytosine. It is the specific arrangement of these bases along a strand of DNA that carries the information necessary to transform a uni-cellular organism into a fully grown adult capable of thinking about its own genetic information. A gene is a section of DNA that codes for a coherent set[1] of potentially overlapping functional products (Gerstein et al., 2007) (proteins or RNA molecules) and it serves as the basic unit of "*heredity*". The different variants of a same gene are called alleles. The human genome is approximately three billion bases long and is packed into bigger units called chromosomes. Moreover, almost 99.9% of the genome is identical between humans, and nearly all of such differences

[1]Most gene definitions actually involve transcription which, for the sake of simplicity, we have not described.

are alterations in a single base pair. These variations are called Single Nucelotide Polymorhpisms (*SNPs*). A locus (loci in plural) is a fixed position in the genome

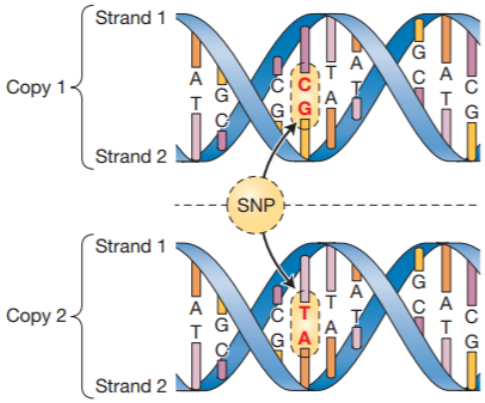


Figure 1.1: Representation of a SNP from Griffiths et al. (2015)

used to locate a specific biological marker such as a SNP. More often than not, SNPs lie in non-coding regions of the genome. These so-called *silent* SNPs are frequently used to study population genetics. However, it is sometimes possible to link a SNP, located in a protein-coding region, to a functional change and thus, to an associated observable alteration. This is the case for several Mendelian disorders such as cystic fibrosis and sickle-cell anemia (Ashley-Koch, Yang, & Olney, 2000).

Population geneticists use allele and genotype frequencies in order to characterize populations. Considering a single locus with two different alleles A and a gives three possible genotypes: homozygotes AA and aa, and heterozygote Aa. Given their respective frequencies f_{AA}, f_{Aa} and f_{aa}, allele frequencies in the population can be expressed in terms of genotype frequencies:

$$\begin{aligned} p &= f_{AA} + \tfrac{1}{2} f_{Aa} \\ q &= f_{aa} + \tfrac{1}{2} f_{Aa} \end{aligned} \quad , \tag{1.1}$$

where p and q are the frequencies of allele A and a, respectively.

In an infinitely large, random-mating population, with no selection, mutation or migration, allele frequencies and genotype frequencies remain constant from one generation to the other as described in Table 1.1. This result is known as the *Hardy-Weinberg equilibrium* (Griffiths et al., 2015).

	Alleles		Genotypes		
	A	a	AA	Aa	aa
freq	p	q	p^2	$2pq$	q^2

Table 1.1: Description of allele and genotype frequencies in a population.

When considering more than one locus, the concept of linkage (dis)equilibrium arises. If an allele at a certain locus is associated with an allele at a different locus more than it would be expected if loci where independent and associated randomly, then those loci are said to be in *linkage disequilibrium*. The amount of linkage disequilibrium present in a population can be estimated from SNP correlations and it can enhance genetic association studies and phenotype prediction algorithms (Sved, McRae, & Visscher, 2008).

Mendel's theory seemed unable to explain the inheritance of continuous or quantitative traits. Fisher et al. (1918) proposed the multi-factorial hypothesis, suggesting that continuous traits are governed by many Mendelian loci, each with a small effect and subject to environmental factors. Since Fisher's work, statistics has played a major role in the study of continuous traits and several phenotype prediction methods have been studied (Falconer & Mackay, 1996). The basic concepts of this field are presented in Section 1.2.

1.2 Quantitative Genetics

Quantitative genetics deals with characters that exhibit continuous variation. These traits, such as height, do not behave in a Mendelian fashion and are usually called quantitative or complex traits. For a comprehensive review of quantitative genetics basics see (Falconer & Mackay, 1996).

Phenotypic values, observed when the character is measured on an individual can be decomposed into their genetic and environmental contributions:

$$P = G + E\,, \tag{1.2}$$

where P is the phenotypic value, G represents genotypic value–i.e. the combined effect of all genes in all the loci which influence the trait–and the environmental deviation E corresponds to all non genetic circumstances that influence the phenotypic value. Since E expresses a deviation from the mean, its expected value is zero. Examples of environmental factors are nutrition, climate, and measurement errors. Genotypic values may be estimated by taking the average phenotypic value of several clones (individuals with the exact same genome) raised *randomly* in different environments. Since $\mathbb{E}(E) = 0$, the average phenotypic value is an unbiased estimator of the genotypic value of that particular genotype, and the concept of population mean refers indistinctly to mean phenotypic value $\bar{P}$ or mean genotypic value $\bar{G}$ of a population.

These notions are conceptually useful, however, parents pass on their genes (alleles) not their complete genotypes to their progeny. In consequence, the concepts of average effect and breeding value arise. The average effect of an allele is defined as the mean deviation from $\bar{P}$ of the individuals who received such allele from one parent, the allele received from the other parent having come at random from the population. The breeding value corresponds to the sum of the average effects of the genes an individual carries. It is sometimes referred to as *additive genotype* since it quantifies the value of an individual based on the expected genotypical values of its progeny.

A simple model for the genotypic value can be expressed as:

$$G = A + D\,, \tag{1.3}$$

where A is the breeding value and the dominance deviation D corresponds to the difference between the genotypic value and the breeding value of an individual. Dominance deviation can be viewed as the result of putting together genes in pairs to make genotypes. If biological architecture was purely additive, the genotypic value would coincide with the sum of the average effects of the genes. When taking into account the interaction between genes at different loci, a term I corresponding to this interaction, often referred to as *epistatic interaction*, is added. Reasonably, this addend is zero if genes act additively betweeen loci.

$$G \;=\; A \;+\; D \;+\; I \tag{1.4}$$

A usual assumption in this context is that these values or deviations are not correlated. In this case, phenotypic variance can be written as the sum of the additive, dominance, epistatic and environmental variance.

$$\sigma_P^2 = \sigma_A^2 + \sigma_D^2 + \sigma_I^2 + \sigma_E^2. \tag{1.5}$$

where σ_A^2, σ_D^2, σ_I^2 and σ_E^2 correspond to additive, dominance, epistatic and environmental variance respectively.

This framework enables the discussion: "heredity vs environment" or "nature vs nurture", which aims to study the relative importance of different sources of variation. This gives rise to the fundamental concept of *heritability*, which is defined as the relative importance of heredity in determining phenotypic values. On one hand, if "heredity" refers to genotypic values, a character is "hereditary" in the sense of being determined by the genotype. This approach to heredity leads to broad sense heritability H^2, also called *degree of genetic determination*.

$$H^2 = \frac{\sigma_G^2}{\sigma_P^2} \tag{1.6}$$

On the other hand, if 'heredity' refers to breeding values, a character is 'hereditary' in the sense of being transmitted from parent to offspring, which leads to narrow sense heritability:

$$h^2 = \frac{\sigma_A^2}{\sigma_P^2}. \tag{1.7}$$

Narrow sense heritability is crucial in prediction problems because it expresses the reliability of the phenotypic value as a guide to the breeding value. Thus, it indicates the responsiveness of a trait to selective breeding, a parameter of utmost importance to plant and animal breeders. Under the assumption that A is uncorrelated with dominance, epistatic and environmental deviations, narrow sense heritability can be viewed as the square of the correlation coefficient between breeding values and phenotypes:

$$r = \frac{Cov(A,P)}{\sigma_A\sigma_P} = \frac{Cov(A, A+D+I+E)}{\sigma_A\sigma_P} = \frac{\sigma_A^2}{\sigma_A\sigma_P} = \frac{\sigma_A}{\sigma_P} = h. \tag{1.8}$$

In consequence, h is the maximum achievable correlation coefficient when fitting an additive linear model.

Chapter 2

Problem formulation

In this chapter, the prediction of complex traits is formulated as a risk minimization problem. Then, the metrics used as loss functions and as measures of predictive ability are presented. That is, the metrics that will be used to *choose* or *fit* the model using training data, and then used to evaluate its performance on test data. Finally, the optimization techniques used to undertake the risk minimization problem are introduced.

2.1 Statistical risk minimization

It is possible to frame the prediction of complex traits as a statistical risk minimization problem (SRM), a standard statistical learning framework (Shalev-Shwartz & Ben-David, 2014). In this context, the samples $x \in \mathcal{R}^p$ are SNP sequences related to phenotypic values $y \in \mathcal{R}$ by the probability distribution $p(x, y)$. The goal is to find a function $\Phi(x)$ that outputs predictions $\hat{y}$ that minimize a certain loss function $\mathcal{L}(\hat{y}, y)$ over the entire distribution (Vapnik, 1991). Thus, finding an optimal solution corresponds to solving the following SRM problem:

$$\Phi^* = \arg\min_{\Phi} \mathbb{E}_{p(x,y)}[\mathcal{L}(y, \Phi(x))] \tag{2.1}$$

In general, the distribution $p(x, y)$ is unknown, making $\mathbb{E}_{p(x,y)}[\mathcal{L}(y, \Phi(x))]$ incomputable. To undertake this obstacle, statistical risk is approximated by an empirical risk, which only contemplates the observed data points: $\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$. Then, using the Law of Large Numbers it is possible to approximate the expectation in equation 2.1 :

$$\mathbb{E}_{p(x,y)}[\mathcal{L}(y, \Phi(x))] \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(y_i, \Phi\left(x_i\right)\right) \tag{2.2}$$

This transforms the statistical risk minimization problem into an empirical risk minimization problem (ERM) (Vapnik, 1991), where the function $\Phi(x)$ is restricted to a class $\mathcal{C}$:

$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\mathbf{y}_i, \Phi\left(\mathbf{x}_i\right)\right) \tag{2.3}$$

As shown in equation 2.3, given the dataset $\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, the choice of the class $\mathcal{C}$ and the loss function $\mathcal{L}(y, \hat{y})$ complete the description of the ERM problem. On one hand, the class function $\mathcal{C}$ is dictated by the model (or parametrization). On the other hand, the choice of the loss function depends on the downstream task. However, minimizing empirical risk may not lead to low statistical risk due to generalization error (Vapnik, 1991). It is thus common to include terms that have an impact on model complexity in the loss function, leading to the Structural risk minimization problem:

$$\Phi^* = \arg\min_{\Phi \in \mathcal{C}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(y_i, \Phi\left(x_i\right)\right) + \mathcal{J}(\Phi), \tag{2.4}$$

where $\mathcal{J}(\Phi)$ is an appropriate regularization or penalty term.

2.2 Metric descriptions

In this section we present several commonly used regression metrics and their limitations, in the context of genome enabled prediction of complex traits.

2.2.1 Pearson correlation coefficient

In the context of genome-enabled prediction of complex traits, the Pearson correlation coefficient (r) between predicted and observed phenotypes is the most popular measure of predictive ability (González-Recio, Rosa, & Gianola, 2014).

$$r_{\hat{\mathbf{y}},\mathbf{y}} = \frac{\operatorname{cov}(\hat{\mathbf{y}}, \mathbf{y})}{\sigma_{\hat{\mathbf{y}}} \sigma_{\mathbf{y}}} \approx \frac{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{\hat{\mathbf{y}}}\right)\left(y_i - \bar{\mathbf{y}}\right)}{\sqrt{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{\hat{\mathbf{y}}}\right)^2} \sqrt{\sum_{i=1}^{n} \left(y_i - \bar{\mathbf{y}}\right)^2}} \tag{2.5}$$

This coefficient is a measure of linear dependence between variables and lies between -1 and 1 . An r close to 0 suggests lack of linear dependence, while an r equal to ± 1 indicates a perfect linear relation between $\hat{y}$ and y.

As a measure of predictive ability, this metric exhibits several limitations (González-Recio et al., 2014; Waldmann, 2019). In particular, being a measure of linear dependence, r is invariant both to scale and bias. In addition, there are radically different point clouds that lead to the same correlation coefficient, as shown in Figure 2.1. However, if the goal is not to predict exact values but rather to rank the individuals, invariance to affine transformations is not an issue, since these transformations preserve order relations. [1] A factor that has contributed to the widespread adoption of the Pearson correlation coefficient is its connection to key

[1] Although affine transformations can also invert the order of a sequence, this does not happen in practice and can easily be corrected.

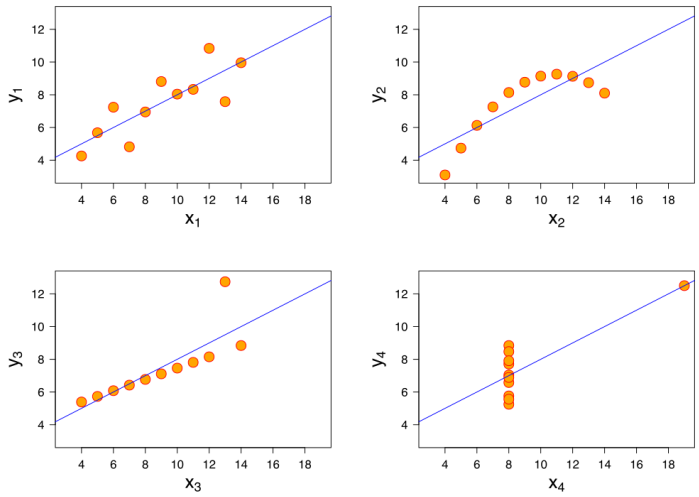


Figure 2.1: Anscombe's quartet: four point clouds with the same correlation coefficient up to 3 decimal places (0.816).

quantitative genetics concepts. In the context of selective breeding, it is common to aim at predicting breeding values. This is due to the fact that, unlike genes, dominance, epistatic and environmental effects are not necessarily passed on from one generation to the next. As showed in Section 1.2, when assuming independece between additive, epistatic and environmental effects, the Pearson correlation coefficient between breeding and phenotypic values corresponds to the narrow sense heritability h of that trait (and in that population). Therefore, when choosing a model to predict breeding values, the highest desirable $r_{\hat{y},y}$ would be h. If the model is linear, h would also be the highest achievable. Higher correlation coefficients would imply that the model responds to non-linear signals in the input, such as latent environmental variables or epistatic effects. It should be noted that this principle is often inapplicable, since h estimates rely on questionable assumptions or simply are unavailable. Moreover, in other applications, where the goal is not to predict breeding values but phenotypic values, capturing non-linear effects on SNPs is desirable. Thus, in such cases, the relation between r and h loses relevance.

2.2.2 Mean squared error

Mean squared error is one of most widely used metrics in regression problems. Being the average squared distance between predicted and observed values it is a non-negative loss.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

2.2.3 Coefficient of determination

The coefficient of determination R^2 is another metric based on squared errors that appears regularly in machine learning.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}$$

As shown in the equation above, R^2 uses a normalized version of the MSE. Thus, it is particularly useful when comparing predictive accuracies in datasets with different variances. Moreover, the quotient in R^2 (mean squared error over data variance) can be seen as the fraction of *unexplained* variance by the model.

An R^2 equal to 1 indicates that predictions perfectly match target values. This corresponds to a MSE of 0. Conversely, a model that always predicts the mean of the target values would get an R^2 equal to 0. Negative values of R^2 can be encountered, since models can be arbitrarily worse than predicting the mean in terms of the mean squared error.

2.3 Choice of loss function and model selection

In this section, we present common metrics used in genome enabled prediction literature to assess model performance. We also note that these are suitable loss functions, and discuss the potential advantages of optimising them directly.

2.3.1 Traditional approach

Except from Bayesian Models and Support Vector Machines [2], most of the models encountered use MSE as a loss function. For instance, all the Ensemble Methods, Linear Regressions and Neural Networks encountered were trained by minimizing the MSE (Abdollahi-Arpanahi, Gianola, & Peñagaricano, 2020; Azodi, McCarren, Roantree, de los Campos, & Shiu, 2019; González-Recio et al., 2014; Grinberg, Orhobor, & King, 2018, 2019; Liu et al., 2019; W. Ma et al., 2018). Some of these models such as Ridge and Lasso Regression add a penalty term to the MSE that impacts model complexity.

However, when selecting and comparing models, the Pearson correlation coefficient between observed and predicted phenotypes is the main metric used (Azodi et al., 2019; González-Recio et al., 2014; Grinberg et al., 2019; Rezende, Nani, & Peñagaricano, 2019; Waldmann, 2019; Yin et al., 2020).
Some authors argue that model selection should not be based solely on r, but also on MSE and/or R^2 (Waldmann, 2019). As mentioned in Section 2.2.1, r is invariant to affine transformations. Thus, a model where $\hat{y}_i = y_i \quad \forall i = 1, 2, .., , N$ and a model where $\hat{y}_i = 3 \times y_i + 10 \quad \forall i = 1, 2, .., , N$ would be indistinguishable from a correlation coefficient standpoint. This example shows that high MSEs do not necessarily correspond to low correlation coefficients.

[2] The models mentioned in this paragraph are explained in Section 4

These considerations raise two question: if the best model is often considered the one with highest correlation coefficient, why not maximize it directly? Is it possible to maximize r while maintaining a low MSE ?

2.3.2 Maximizing Pearson correlation

As described in the Section 2.2.1, r is extensively used as a metric to compare models but not as a loss function in the ERM problem. Several issues need to be addressed in order to use the Pearson correlation coefficient as a part of the loss function. Firstly, the ERM formulation (and most optimization libraries) are designed to minimize targets. However, maximizing r is equivalent to minimizing $-r$. Thus, maximizing the correlation coefficient is equivalent to solving the ERM problem with Negative Pearson Correlation as the loss function:

$$L(\hat{y}, y) = -r_{\hat{y},y} = -\frac{\sum_{i=1}^{n}\left(\hat{y}_i - \bar{\hat{y}}\right)(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}\left(\hat{y}_i - \bar{\hat{y}}\right)^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}. \tag{2.6}$$

Secondly, as discussed in Section 2.2.1, invariance to affine transformations makes this loss function ill-suited for prediction purposes. A model could maximize r by learning to predict an affine transformation of the targets: $\hat{y} = ay + b$, leading to a high MSE. Nonetheless, after the optimization, it would be possible to estimate the parameters a and b of this linear transformation. If the linear dependence (correlation) between the observed and predicted phenotypes is high, undoing the estimated affine transformation should yield accurate phenotype predictions. This method was implemented and tested in Graph Neural Networks, and its outcome is discussed in section 7.7.8.

2.4 Optimization

In this Section we describe some commonly used optimization methods for differentiable, non-convex empirical risk minimization problems. We focus on popular techniques in the context of neural networks and deep-learning. Also, a novel optimisation framework which stems from a modified optimisation problem is introduced.

2.4.1 Gradient descent

Most of the models explored use Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951) to undertake the structural risk minimization problem. The term *stochastic* reflects the fact that batches of data are used to *estimate* the gradient of the loss function with respect to the weights of the model. In this context, we consider the per-batch average loss function:

$$L(\mathbf{W}) = \frac{1}{Q}\sum_{q=1}^{Q}\mathcal{L}\left(y_q, \Phi\left(x_q; \mathbf{W}\right)\right)$$

where Q is the batch size and the dependence of Φ and L on the model weights $\mathbf{W}$ (or trainable parameters) is made explicit. Since the weights are updated at each step of the algorithm, the subindex t is added to represent the step number. To simplify the notation, we will refer to the gradient of the loss function with respect to model weights, evaluated at W_t, as dW_t:

$$dWt = \nabla_{\mathbf{W}} L(\mathbf{W_t})$$

In its most simple version, the update rule for SGD is the following:

$$W_{t+1} = W_t - \eta \, dW_t$$

where η is the learning rate, W_{t+1} and W_t correspond to the weights of the model at step $t+1$ and t respectively and dW_t represents the gradient of the loss function with respect to the weights evaluated at W_t.
A variation of this rule, which often has better convergence rates, is the *momentum update* (Rumelhart, Hinton, & Williams, 1986):

$$\begin{aligned} v_{t+1} &= \mu \, v_t - \eta \, dW_t \\ W_{t+1} &= W_t + v_{t+1} \end{aligned}$$

where μ, usually referred to as *momentum*, is a hyperparameter and v_t denotes the update term at step t. By computing the update as a linear combination of the gradient and the previous update (weighted moving average), SGD with momentum discourages drastic changes in the update term, reducing oscillations. The bigger the momentum μ, the more the oscillations are dampened.

A slightly different version of the momentum update is the so-called Nesterov momentum (Nesterov, 1983). The key aspect of Nesterov momentum is that the gradient is evaluated at a *lookahead* approximation of the weights. In the momentum update rule, the momentum μ is usually close to 1. Thus, W_{t+1} can be approximated by $W_t + \mu v_t$. Evaluating the gradient of the loss function at this approximation of the weights at time $t + 1$ increases the *responsiveness* of the update rule (Nesterov, 1983).

$$\begin{aligned} \hat{W}_t &= W_t + \mu \, v_t \\ v_{t+1} &= \mu \, v_t - \eta d\hat{W}_t \\ W_{t+1} &= W_t + v_{t+1} \end{aligned}$$

where $d\hat{W}_t = \nabla_{\mathbf{W}} L(W_t + \mu v_t)$. A simplified, two-dimensional representation of the Momentum and Nesterov Momentum update rules is shown in Figure 2.2.

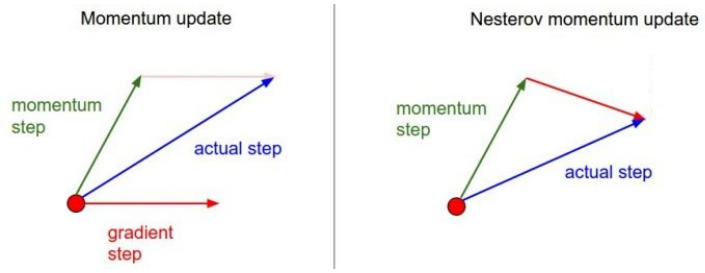


Figure 2.2: Graphical representation of Momentum and Nestrov Momentum update rules from *http://cs231n.stanford.edu/*.

The three approaches discussed use constant learning rates for all parameters. However, some methods adaptively tune the learning rates and do so per parameter. One of these update rules is AdaGrad (Duchi, Hazan, & Singer, 2011) (short for Adaptive Gradient):

$$G_t = \sum_{i=1}^{t} dW_i \cdot dW_i^T$$
$$W_{t+1} = W_t - \eta \ \mathrm{diag}(G)^{-1/2} \, dW_t$$

where the diagonal of the matrix G accumulates squared gradients and is used to normalize the update term element-wise. In Adagrad, the effective learning rates are monotonically decreasing. However, the rate at which they decrease can be different for each parameter.

Finally, one of the most popular update rules with adaptive learning rate is Adam (Kingma & Ba, 2014):

$$m_{t+1} = \beta_1 \, m_t + (1 - \beta_1) dW_t$$
$$v_{t+1} = \beta_2 \, v_t + (1 - \beta_2)(dW_t \odot dW_t)$$
$$W_{t+1} = W_t - \eta \frac{m_{t+1}}{\sqrt{v_{t+1}}}$$

where $\odot$ denotes element-wise multiplication. The variable v, also called second moment, plays the same role as G in Adagrad (i.e: accumulate square gradients). The term m, known as first moment, is a weighted average of the previous m and the current gradient (weighted moving average). Thus m represents a smoothed version of dW_t. The real numbers β_1 and β_2 are the decay rates of the moving averages m and v. In contrast to AdaGrad, in Adam, the sum of square gradients is *leaky*, enabling the increase of effective learning rates. [3].

[3]The complete Adam update rule includes bias-correction terms for the first and second moment estimates.

2.4.2 Sharpness aware minimization

Flatness of loss function's minima and its long known (Hinton & van Camp, n.d.; Hochreiter & Schmidhuber, 1995) connection with generalisation has regained interest in the context of deep over-parametrized neural networks, due to recent empirical results (Jiang, Neyshabur, Mobahi, Krishnan, & Bengio, 2019; Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang, 2016; Li, Xu, Taylor, Studer, & Goldstein, 2017) and theoretical bounds (Dziugaite & Roy, 2017; Wei & Ma, 2019). Since it is accepted that flatter minima generalise better, recent methods have tried to incorporate sharpness in the minimization process (Chaudhari et al., 2019; Foret, Kleiner, Mobahi, & Neyshabur, 2020). Sharpness Aware Minimization (SAM)(Foret et al., 2020) defines the objective function L^{SAM} as:

$$L^{SAM}(W) \triangleq \max_{\|\epsilon\|_p \leq \rho} L(W + \epsilon),$$

where $L(W)$ is the loss evaluated with weights W. This is the maximum of the loss in a ball with radius ρ on the parameter space. The proposed algorithm to solve the resulting minimax problem consists of performing a gradient ascent step to find the maximum loss in a radius ρ ball of current weights, and then calculate gradients on that point to perform the gradient descent step to update weights. Foret et al. (2020) give a formal derivation of the method.

Chapter 3

Datasets used in this work

Since the field lacks a single dataset used for benchmarking, we chose four datasets of different characteristics, two of which are publicly available. These datasets are diverse in terms of population structure, trait complexity, number of samples and number of SNPs. In this Chapter we describe each of these datasets. We then describe how that data is processed prior to training on Section 3.2, and perform an exploratory data analysis on Section 3.3. A summary of the main characteristics of each dataset can be found in Table 3.1 .

Dataset	Samples (N)	Markers (p)	N/p	Phenotype(s)	# Envs.
Yeast	$1,008$	$11,623$	0.087	Growth	4
Jersey	$1,569$	$107,371$	0.015	Sire Conception Rate	1
Wheat	599	$1,447$	0.41	Grain Yield	4
				Milk Fat Percentage	
Holstein	$5,024$	$42,551$	0.11	Milk Yield	1
				Somatic Cell Score	

Table 3.1: Datasets' summary.

3.1 Datasets

3.1.1 Yeast growth

The yeast dataset (Bloom, Ehrenreich, Loo, Lite, & Kruglyak, 2013) consists of 1008 samples of yeast strains which were obtained as the cross (meiosis[1]) between a laboratory and a wine strain. Both parent strains correspond to the species *Saccharomyces cerevisiae* and differed in only 0.5% of their markers. The phenotype of interest is yeast growth and it was measured in 48 different environments. We study four environments: Lactate, Lactose, Xylose, Sorbitol.

[1]process where a single cell divides twice to produce four cells containing half the original amount of genetic information

3.1.2 Jersey bull fertility

The Jersery Bull Fertility dataset consists of 1.569 bulls whose Sire Conception Rates (SCR) were evaluated on 29 different occasions ranging from 2008 to 2018.

The SCR is a phenotypic trait defined as the expected difference between the conception rate of a bull compared with the mean of the rest of the population. The conception rate is defined as the amount of successful inseminations as a fraction of the total inseminations attempted (performed within an evaluation instance).

A single bull may have multiple evaluations of its SCR; in this case, we take a weighted mean of all the instances available. Each SCR measure is weighted by its reliability (REL) value, which is calculated using the number of total inseminations n as $\text{REL} = 100[n/(n + 260)]$ (n ranges from 200 to 26.000 in some cases). Each SCR record has its own associated REL value which is also provided by the CDCB.

The genotypic data available consists in $107,371$ markers (SNPs) for each individual. Markers that mapped to sex chromosomes, had minor allelic frequencies below 1%, or had a call rate (rate of successful measurements across the dataset) of less than 90% were removed, resulting in a total of $95,434$ markers (Rezende et al., 2019).

3.1.3 Wheat Yield

This dataset (Crossa et al., 2014) consists of 599 wheat strains developed by the CIMMYT Global Wheat Breeding program[2]. These strains were grown in four different environments, measuring the grain yield (amount of crop grown per unit area of land) as the phenotypic trait of interest.

The original dataset consists of 1447 markers. As with the previous datasets, a stage of preprocessing was applied: markers with an allele frequency lower than 5% or greater than 95% were removed. This resulted in 1279 total markers.

3.1.4 German Holstein

This dataset describes a German Holstein population of 5024 bulls. The genomic sequences, after being filtered for quality control, consist of $42,551$ SNPs.

Each bull's estimated breeding value in three different traits was measured[3]. These traits are all closely related to milk production: milk fat percentage (MFP), somatic cell score (SCS) and milk yield (MY). It is worth noting that each trait represents a different underlying genetic architecture.

The SCS is governed by many small effect loci, MY is determined by a few moderate effect loci and many small effect loci and MFP is composed of a few major genes and a large number of loci with small effects (Z. Zhang et al., 2015).

[2]https://www.cimmyt.org/

[3]This estimation was done measuring the phenotypic value of the traits in the progeny of each bull.

3.2 Data preparation

Individuals of the same species share most of their DNA. Reasonably, in order to predict traits that vary in a population, it is necessary to analyze only the sections of the genome that differ from individual to individual. As explained in Section 1, most of the variability in the genome is attributable to SNPs. Moreover, most of SNPs are biallelic, which means that only two variants are observed in the population.

In this work, most of the organisms studied are diploid, that is, they have two possibly different copies of each gene (one from each parent). Therefore, the genetic content of each locus can be described by an element of the following ternary alphabet: $\{AA,\ Aa,\ aa\}$. Usually, a represents the most frequent base present in that locus while A represents the rare variant.

3.2.1 Encoding

All predictive models operate on numerical inputs, which implies that the alphabet {AA, Aa, aa} needs to be adapted to a numeric form.

In order to preserve the categorical nature of these biological variables, it is reasonable to use *One Hot Encoding (OHE)*. This codification scheme maps each SNP to a vector: $AA \to \{1,0,0\}$, $Aa \to \{0,1,0\}$ and $aa \to \{0,0,1\}$. In practice, one category is erased, since it is redundant given the other two. The resulting codification is $AA \to \{1,0\}$, $Aa \to \{0,1\}$ and $aa \to \{0,0\}$ (see Figure 3.1).

$$X = \begin{bmatrix} AA & aA & \dots & AA \\ aA & aa & \dots & Aa \\ \vdots & \vdots & \ddots & \vdots \\ AA & Aa & \dots & aa \end{bmatrix} \xrightarrow{OHE} X_{OHE} = \begin{bmatrix} 1 & 0 & 0 & 1 & \dots & 1 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & 1 & \dots & 0 & 0 \end{bmatrix}$$

Figure 3.1: Example of One Hot encoded input matrix. The OHE encoded matrix has twice as much columns as the original input matrix.

Another possibility is to perform an additive codification of SNPs: $AA \to 2$, $Aa \to 1$ and $aa \to 0$ (see Figure 3.2). The underlying assumption behind this codification is that the effects of SNPs at a locus are additive. Thus, the more rare SNPs a genotype has, the further away is its breeding value from the population mean. This is the standard codification used in Quantitative Genetics (Falconer & Mackay, 1996).

OHE may reflect the categorical nature of these variables more faithfully. However, it doubles the dimensionality of the input signal. The empirical consequences of this trade-off are studied in Chapter 4.4. In haploid organisms, the alphabet has two elements : $\{a, A\}$. In this case, additive and one hot encoding are equivalent.

$$X = \begin{bmatrix} AA & aA & \dots & aa & AA \\ aA & aa & \dots & Aa & Aa \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ AA & Aa & \dots & Aa & aa \end{bmatrix} \xrightarrow{Add.} X_{Add} = \begin{bmatrix} 2 & 1 & \dots & 0 & 2 \\ 1 & 0 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 2 & 1 & \dots & 1 & 0 \end{bmatrix}$$

Figure 3.2: Additive encoding example.

3.2.2 Imputation

Genotyping techniques are not perfect, therefore, some markers are missing from the datasets. To solve this problem, the following imputation techniques were employed: sample deletion, marker deletion, mean imputation and mode imputation. The choice of imputation was done taking into account the dimensions of the datasets. For example, in the Jersey data set, every sequence has at least one missing SNP. In that case, sample deletion is nonsensical.

3.3 Brief Exploratory Data Analysis

To gain more insight into each datasets characteristics, we conducted an exploratory analysis. This analysis consisted in exploring the input signal's structure, the target variable's distribution, and the correlation between different phenotypes or environments (belonging to the same dataset).

3.3.1 Yeast

Figure 3.3 shows the correlation between the target values across four different environments. These environments have relatively high pairwise correlations compared to others in the dataset. This is because they are sugars which encourage yeast's growth. The pairwise correlation between environments is relevant for multi-task learning approaches, which will be explored in Chapter 6.

The target phenotype closely resembles a normal distribution on most environments, as seen in Figure 3.4.

The sequences of raw data were post-processed into $30,594$ high confidence SNPs which were uniformly sampled to obtain the final $11,623$ binary markers. These were coded as follows: 1 if the sequence variation came from the wine strain and 0 if it came from the laboratory strain.

As seen in Figure 3.5, the yeast marker sequence has large, constant chunks. This can be attributed to the fact that individuals in this dataset come from the same two parent strains and that nearby markers are likely to be inherited together.

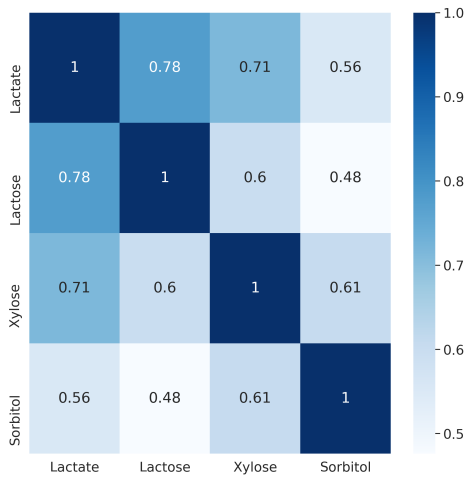
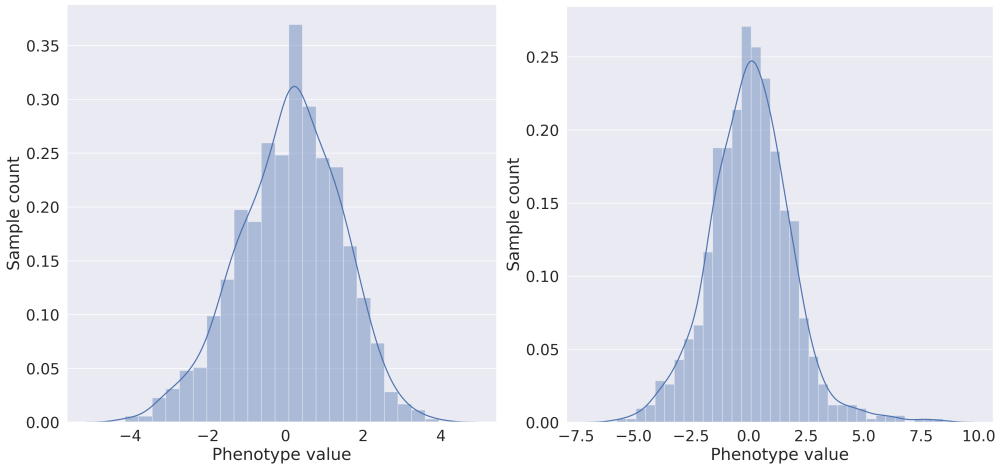
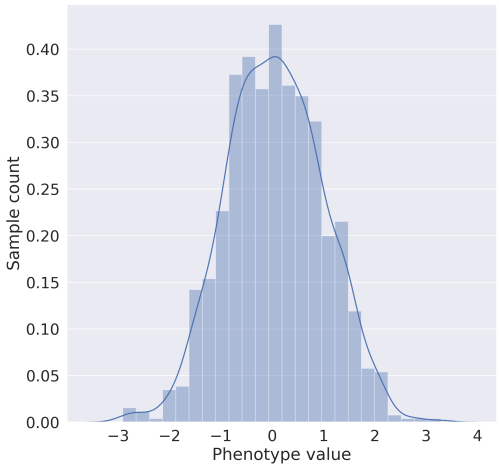


Figure 3.3: Correlation between yeast growth in four environments.

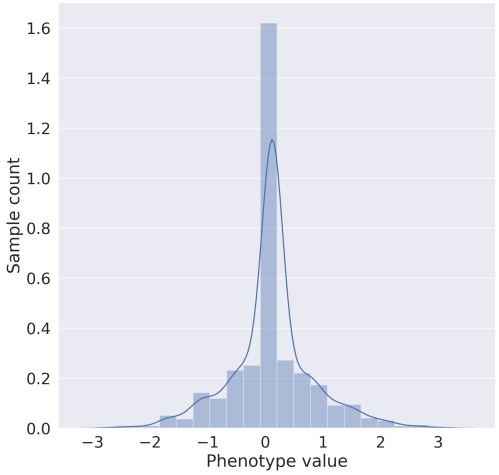


(a) Lactate

(b) Lactose



(c) Xylose



(d) Sorbitol

Figure 3.4: Histograms of four centered Yeast phenotypes. The blue lines represent the Kernel Density Estimation (KDE) plot.

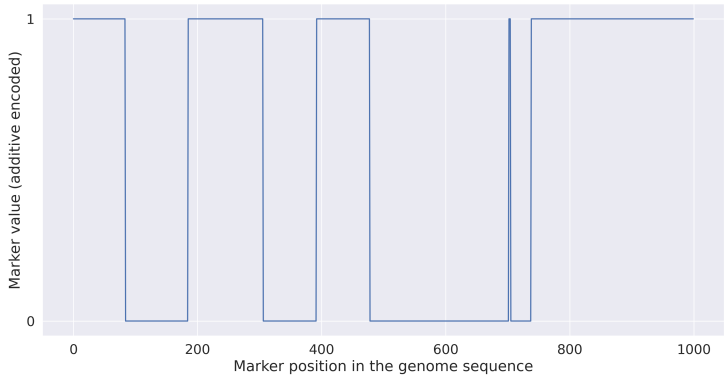


Figure 3.5: Plot of a section of Yeast strain's markers.

3.3.2 Jersey

Figure 3.6 shows the histogram of target phenotype values. Similar to yeast, the phenotype also resembles a normal distribution. On the other hand, Figure 3.7 shows the first 100 markers of a Jersey Bull's genotype. The structure of the input genotype in this dataset differs significantly from yeast's, with no easily-identifiable pattern.

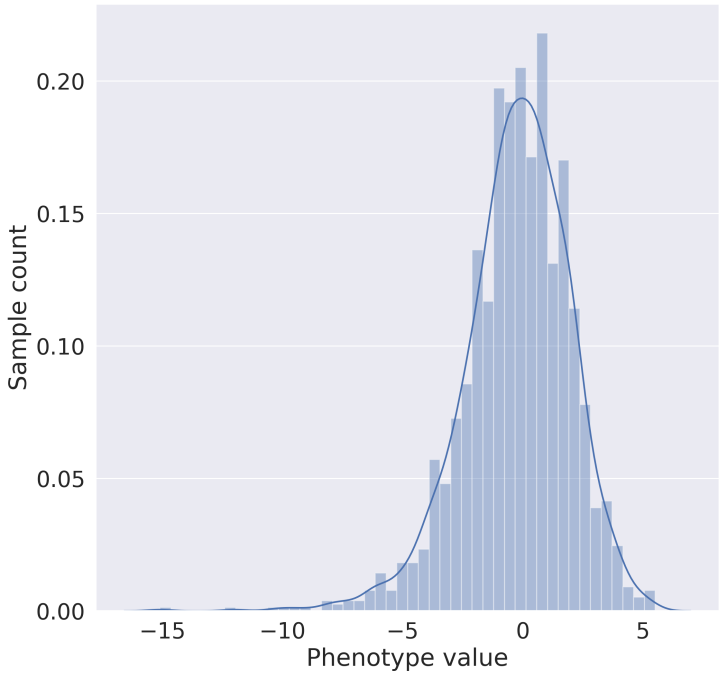


Figure 3.6: Histogram of Jersey centered phenotype.

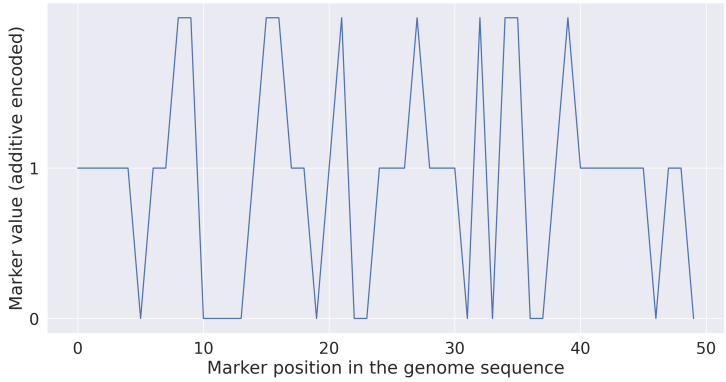


Figure 3.7: First 100 markers of a Jersey sample.

3.3.3 Wheat

Figure 3.8 shows the pairwise correlations between the target phenotypes in different environments. The environments exhibit varying degrees of correlation, from uncorrelated (-0.02) to weak (-0.12, -0.19) and medium (0.39, 0.41, 0.66) correlation values. The phenotype distribution also qualitatively ressembles a normal distribution in all four environments, as seen in Figure 3.9.

In terms of signal structure, Wheat's genotypes resemble Jersey's, in the sense that no clear spatial correlations can be distinguished. This is shown in Figure 3.10.

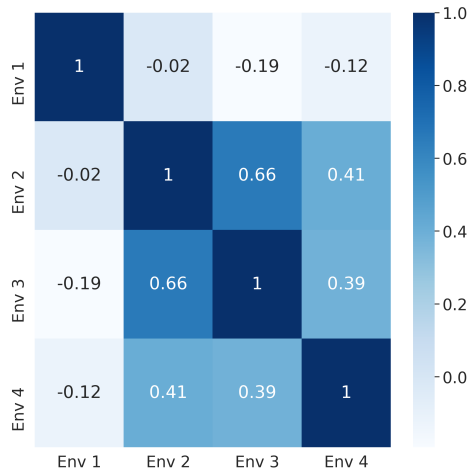
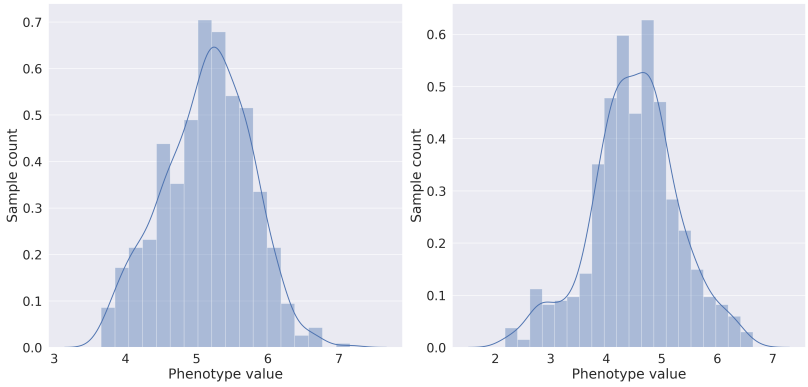
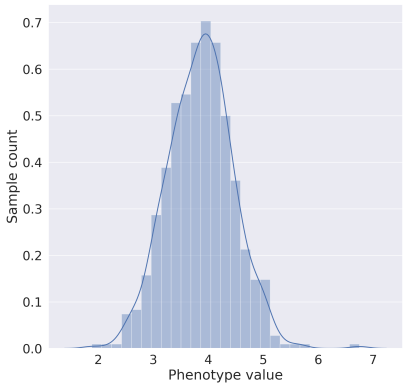


Figure 3.8: Correlation between Wheat Yield in four environments.

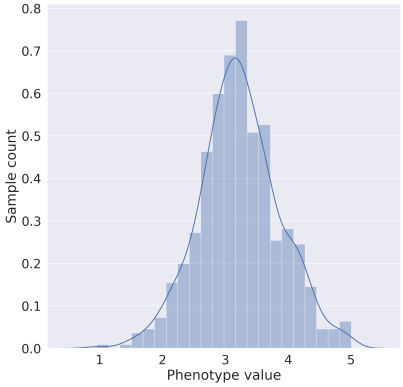


(a) Environment 1

(b) Environment 2



(c) Environment 3



(d) Environment 3

Figure 3.9: Histograms of Wheat Yield phenotypes in each environment.

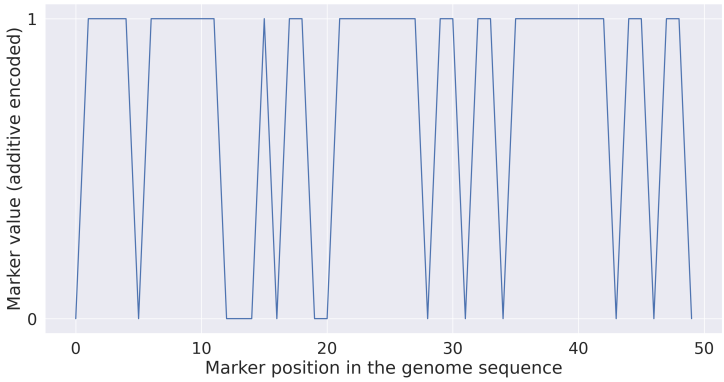


Figure 3.10: First 50 markers of a Wheat sample.

3.3.4 Holstein

As shown in Figure 3.11 MY and MFP are strongly negatively correlated, while the other two pairwise correlations (MY-SCS, MFP-SCS) are practically null.

Once again, all three target phenotypes exhibit normal distributions, as shown in Figure 3.12.

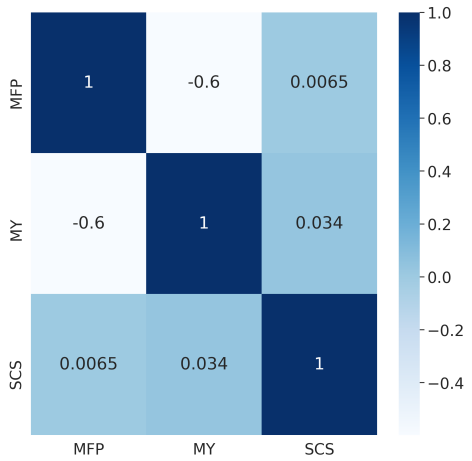
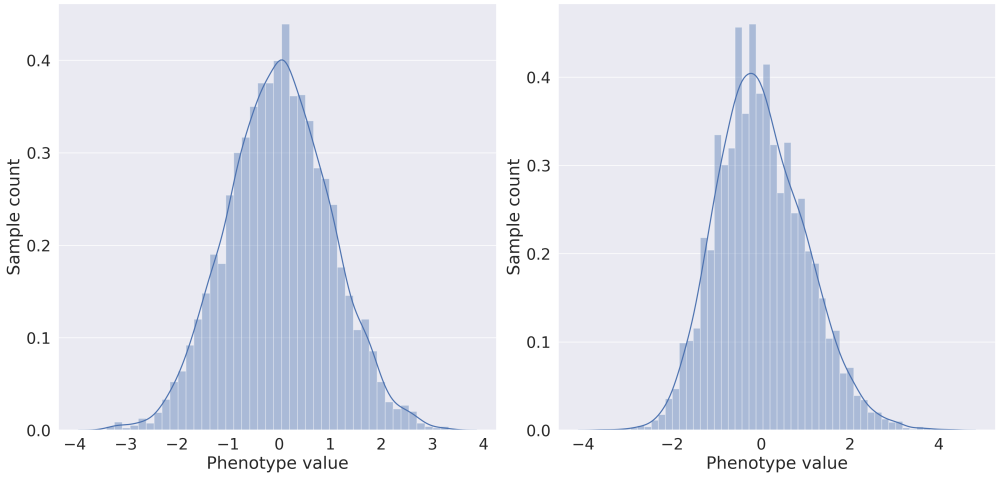
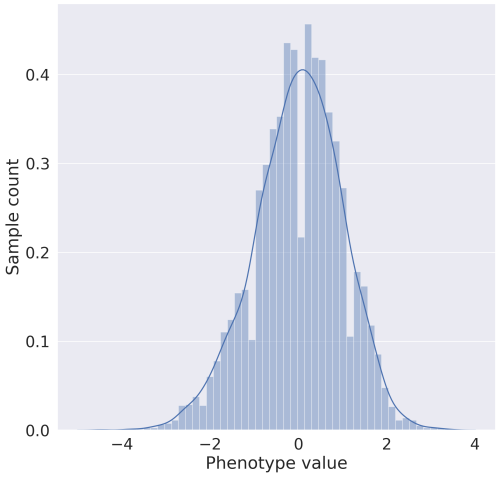


Figure 3.11: Correlation between Holstein traits.



(a) MFP

(b) MY



(c) SCS

Figure 3.12: Histograms of Holstein centered phenotypes.

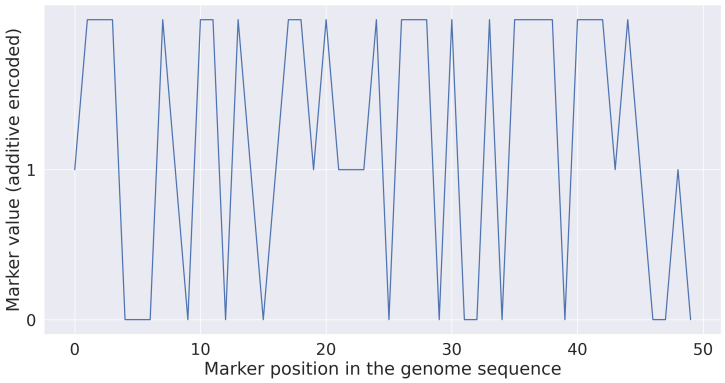


Figure 3.13: First 50 markers of a Holstein sample.

This page intentionally left blank.

Chapter 4

Review of predictive models in genomics

A plethora of machine learning and statistical models have been applied in the context of genomic prediction. Unsurprisingly, no method outperforms all others across different species, traits and populations (Azodi et al., 2019). Limitations for comparing between models found in literature stem from the fact that several datasets are used, and that the field lacks a widely adopted dataset for model benchmarking. Population structure, complexity of traits, number of samples and SNPs all present significant differences between datasets, factors which undeniably affect model performance.

Furthermore, most publications include and compare a reduced set of models, as shown in Figure 4.1. As can be seen, most models are linear and the Bayesian approach draws significant attention. Furthermore, Neural Networks received little interest up to 2018, although they have gained attention recently (Abdollahi-Arpanahi et al., 2020; Azodi et al., 2019; Pérez-Enciso & Zingaretti, 2019).

Several studies comparing different models exist (Abdollahi-Arpanahi et al., 2020; Azodi et al., 2019; González-Recio et al., 2014; Grinberg et al., 2018). Most methods are classical in the realm of machine learning and statistical estimation. A detailed formulation of *Support Vector Regression* and *Ensemble Methods* is included in appendix C. We outline the most popular methods below, with the aim of introducing domain specific terms and references.

4.1 Linear regressions

Since Fisher's seminal work (Fisher et al., 1918), genetic effects have been partitioned into linear and non-linear effects. Substantial efforts have been devoted to describe the former on classical Quantitative Genetics literature (Falconer & Mackay, 1996).

Even though the cost of high density genotyping is rapidly decreasing (Akdemir & Isidro-Sánchez, 2019), the number of markers far exceeds the number of individuals in most datasets. Therefore, regularization, often referred to as *shrinkage*, plays a major role in determining marker effects.

In that sense, the community has embraced Bayesian Linear Regressions as

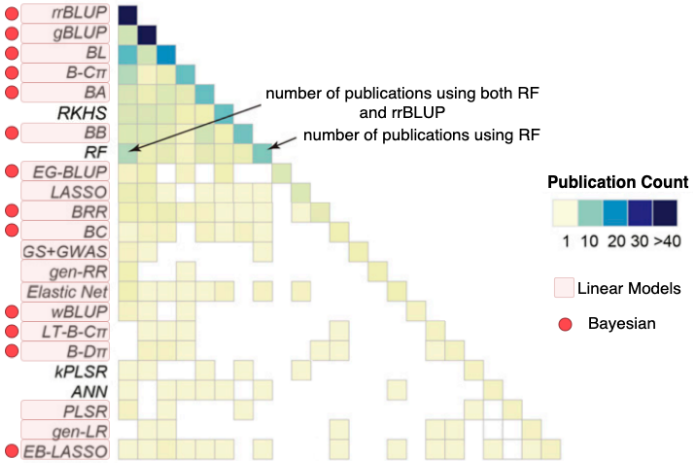


Figure 4.1: Confusion matrix showcasing the number of publications including different models, ordered by total number of publications (most popular models at the top), from 91 publications published between 2012-2018. Figure adapted from (Azodi et al., 2019).

the de-facto standard for genomic prediction for decades (Gianola, 2013; Gianola & Fernando, 1986). As shown on Figure 4.1 these models still draw significant attention on recent publications.

In the following Section we describe some of the most commonly used models, for a thorough review of bayesian methods see (Gianola, 2013; Zaabza, Gara, & Rekik, 2017).

Using a linear model phenotypes can be expressed as

$$y = x\beta + E, \tag{4.1}$$

where $y \in \mathbb{R}$ represents a phenotype and $x \in R^p$ a genotype consisting of p SNPs, usually with additive coding (0, 1, 2 for diploid organisms - presented in Chapter 3, Section 3.2.1). $x\beta$ represents the additive term A and the residual term E encompasses dominance (D), epistatic (I) and environmental deviations. All of these terms were introduced and described Chapter 1.

4.1.1 Bayesian Linear Regressions

In Bayesian Linear Regressions, β is estimated by maximizing the posterior probability, i.e Maximum A Posteriori (MAP) estimation:

$$\beta = \underset{\beta}{\operatorname{argmax}}\, p\left(\beta \mid x, y\right),$$

Where $p\left(\beta \mid x, y\right)$ is the posterior distribution, which using bayes theorem can be expressed as

$$p(\beta \mid x, y) = p(y \mid x, \beta)p(\beta),$$

where $p(\beta)$ is the prior probability distribution of β.

Posterior distributions are sampled using Markov Chain Monte-Carlo methods (Sorensen & Gianola, 2007), where a markov chain with the desired stationary distribution is created. Maximum a Posteriori estimators are then obtained for all parameters.

The prior for the residual E is gaussian for all models, although results are not always consistent with this assumption.

$$\mathbf{E} \sim \mathcal{N}\left(0, \sigma_{\mathrm{E}}^2\right), \tag{4.2}$$

where σ_{E}^2 is the environmental effects' variance.

Bayesian linear models commonly used in genomic prediction differ *only* on the prior imposed to weight vector β, which is also called *marker effects*. We present the most popular ones below.

Marker Effect Priors.

Each coefficient in β is usually modelled as a random variable with normal distribution and zero mean:

$$\beta_j|\sigma_j \sim \mathrm{N}\left(0, \sigma_j^2\right). \tag{4.3}$$

Predictors derived from this model with σ_j fixed for all markers are commonly referred to as Best Linear Unbiased Predictors (BLUP or rrBLUP) in the context of genome-enabled prediction literature.

For Bayes A and C, σ_j is has an inverse chi-squared distribution:

$$\sigma_{\beta_j}^2 \sim \chi^{-2}(\nu, S), \tag{4.4}$$

where ν are the degrees of freedom and S the scale parameter.

On Bayes A marker variance is sampled independently for each marker (iid) while in Bayes C it is sampled only once, i.e. all marker effects have the same variance.

Bayes B induces sparsity by adding a hierarchical prior of no marker association:

$$\sigma_{\beta_j}^2|\pi \sim \begin{cases} 0 \text{ with probability } \pi \\ \chi^{-2}(\nu, S) \text{ with probability } (1-\pi) \end{cases} \tag{4.5}$$

where the fraction of non-zero efects π is assumed to be uniformly distributed,

$$\pi \sim \mathcal{U}(0, 1). \tag{4.6}$$

GBLUP uses the genomic relationship matrix G, which can be derived as an estimation of pedigree from markers, to set marker variance components:

$$\beta|\sigma_\beta, G \sim \mathrm{N}\left(0, G\sigma_\beta^2\right) \tag{4.7}$$

$$G = \frac{X'X}{\sum_{i=1} var\left(SNP_i\right)}, \tag{4.8}$$

Bayesian Lasso, alternatively, uses a Laplacian prior for β:

$$\beta|\lambda \sim \mathcal{L}(0, \lambda) \tag{4.9}$$

Cross validation is commonly used to set priors and hyperparameters for each dataset.

4.1.2 Penalised Linear Regressions

As explained in Section 2.1, it is common to include penalty terms in the loss function to reduce model complexity. *Ridge* (Hoerl & Kennard, 1970) and *Lasso* (Tibshirani, 1996) Regression are linear models where a penalty term is included in a MSE loss function. In Ridge, the sum of the squares of the marker effects is added to the loss function:

$$L_{Ridge} = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i\beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where N is the number of samples and the hyper-parameter $\lambda \in \mathcal{R}$ affects the shrinkage- or regularization- intensity. In Lasso, the sum of the magnitudes of marker effects is added:

$$L_{Lasso} = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i\beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Both Ridge and Lasso reduce model complexity by shrinking marker effects (i.e: favouring small weights). However, Lasso also induces sparsity, which can be viewed as feature selection, as explained in (Tibshirani, 1996). Bayesian Linear Regressions and Penalised Linear Regressions are closely related. In fact, for specific values of λ, Ridge and Lasso Regressions are equivalent to Bayesian Linear Regressions, with a Gaussian and Laplacian prior over marker weights respectively (Gianola, 2013).

In Elastic Net (Zou & Hastie, 2003), the penalty terms associated to Ridge and Lasso regression are both added:

$$L_{ElasticNet} = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i\beta)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

where λ_1 and λ_2 are scalars that control shrinkage intensity. By setting λ_1 or λ_2 to zero Ridge or Lasso Regressions are recovered.

4.2 Support Vector Regression

Kernel methods have gained popularity in quantitative genetics due to their potential to capture non-linear relationships (Morota & Gianola, 2014). A detailed explanation of SVR can be found in Section C.1. In Support Vector Regression

(SVR) predictions are constrained [1] to deviate from the ground truth by a value no greater than a user-specified constant ϵ:

$$|y_i - \Phi(x_i)| \leq \epsilon, \forall i$$

In its most simple formulation, the function Φ is a linear model (i.e: $\Phi(x) = x\,\beta + b$) and a regularization penalty is imposed to the weights:

$$\beta^T \beta \leq C$$

where $C \in \mathcal{R}$ is a hyperparameter. As explained in appendix C.1, the name *Support Vector Regressor* stems from the fact that the function Φ is completely determined by a subset of input vectors named support vectors.

To enable more complex, non-linear predictors, inputs are transformed into a higher-dimensional (even infinite-dimensional) space using non-linear transformations. However, these non-linear transformations are not applied directly. Instead, only the inner-product between transformed samples is computed. As explained in appendix C.1, Kernels (i.e: continuous symmetric positive semidefinite function) are used to compute these inner-products. Thus, the choice of the kernel determines the underlying transformation.

The most commonly used kernel, introduced in the genomic prediction context by Gianola et al. (Gianola & Van Kaam, 2008), is gaussian radial basis functions (RBF). A number of other kernels have also been proposed and studied, such as the *t-kernel* (Tusell, Pérez-Rodríguez, Forni, Wu, & Gianola, 2013) and the *diffusion kernel* (Morota, Koyama, Rosa, Weigel, & Gianola, 2013). However, none has gained widespread adoption due to negligible gains in performance on real data (Morota & Gianola, 2014).

Support Vector Machines can be recovered as a particular case of Reproducing Kernel Hilbert Space Regression Models (RKHS) (de los Campos, Gianola, & Rosa, 2009) by using the Hinge Loss, as defined in equation 4.10.

$$\mathcal{L}_\epsilon(\Phi(x), y) = \begin{cases} 0, & \text{if } |\Phi(x) - y| < \epsilon \\ |\Phi(x) - y| - \epsilon, & \text{otherwise} \end{cases} \tag{4.10}$$

4.3 Tree Ensemble methods

Model ensembles consist of combining different models (weak learners) in order to obtain a better performance than any of the individual models'. Thus, ensembling can be thought of as a way of compensating for sub-optimal models with extra computation. Although the ensemble model represents a single function, it does not necessarily belong to the same function class as the models from which its built. For example, a simple ensemble model could be a weighted average between the predictions of a regularized linear regressor and a support vector regressor.

[1] As shown in appendix C.1, slack variables are later introduced to loosen this constraint.

In tree-based ensembling, the weak learners consist of decision trees (DTs). DTs are a non-parametric supervised learning algorithm which predicts the target variable by learning simple decision rules from the training data.

Although not as popular as the aforementioned, there are several works which use tree-based ensemble methods, namely random forests (Botta, Louppe, Geurts, & Wehenkel, 2014; Goldstein, Hubbard, Cutler, & Barcellos, 2010; Holliday, Wang, & Aitken, 2012) and gradient boosting machines (González-Recio, Jiménez-Montero, & Alenda, 2013). These are also capable of modeling non-linear relationships, but shallower decision trees have been found to perform more favourably on most datasets (Azodi et al., 2019; Goldstein et al., 2010). Random Forest and Gradient Boosting, which are used in this work, are explained in detail in appendix C.1.

4.4 Results and Discussion

Hetereogeneity in heritability, genetic architecture, marker and sample dimensions among datasets enhances model performance analysis. Hyperparameter searches and fine tuning were conducted for each dataset using randomized five-fold cross-validation. Hyperparameters, grid serch results, predictions and metrics can be found on `https://www.comet.ml/dna-i` for each experiment. It should be noted that the main purpose of these experiments is the establishment of baseline results and not an in-depth analysis of traditional models and their hyperparameters.

Due to high variability in training and testing sets and small sample sizes, which results in fluctuating errors that are highly dependent on training-test data splits, each experiment was repeated 10 times using different random splits. Reported metrics correspond to the average of those 10 splits. In most cases, results per split are also included, so as to illustrate and compare performance dependency on splits between models.

4.4.1 Jersey Bull Fertility

On average, all models outperformed the best results from (Rezende et al., 2019). On a side note, this may be viewed as yet another example of the *"Bitter Lesson"* (Sutton, 2019) that generalist approaches leveraging computation *can* outperform those more reliant on domain knowledge.

As already noted, the low volume of data, together with high environmental noise leads to results which vary greatly depending on the train-test split. To illustrate this, results from all 10 splits are included in Figure 4.2. Although some models are more affected than others, this dataset provides an example of such behaviour.

Furthermore, Additive encoding led to higher mean predictive correlations than One Hot encoding in all models. The models trained on One Hot encoded data also exhibit greater variance in predictive correlation. As mentioned in Section 3.2.1, the aim of using One Hot Encoding is to preserve the categorical nature of the variables and enable the model to capture non additive effects. However, OHE comes with the downside of doubling the dimensionality of the input. Moreover,

in One Hot Encoded samples, the one-to-one correspondence between features and biological markers is lost. In other words, the information regarding which pairs of binary variables in OHE correspond to the same marker is lost. This distorts the input's structure and possibly hinders the learning process.

The additivity of the trait's architecture combined with the aforementioned downsides of OHE may explain why this coding scheme is outperformed by additive encoding.

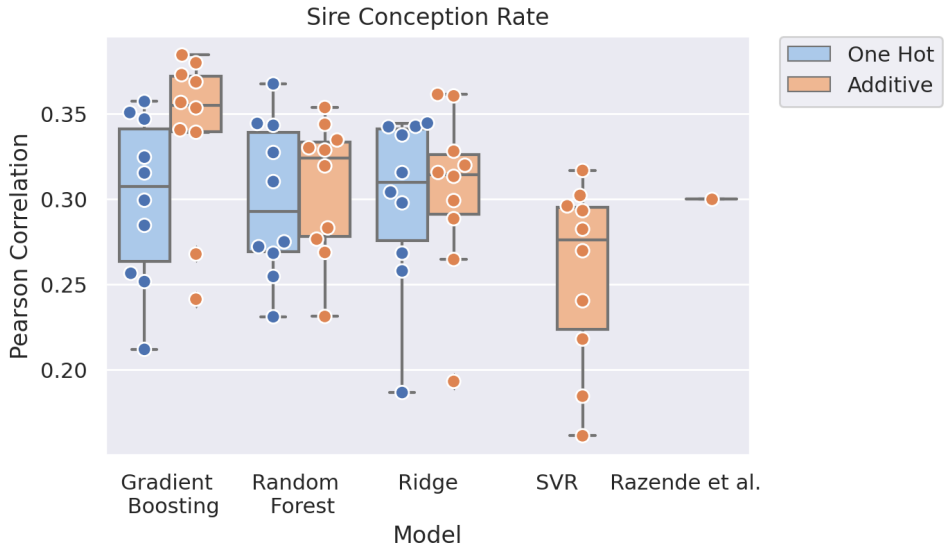
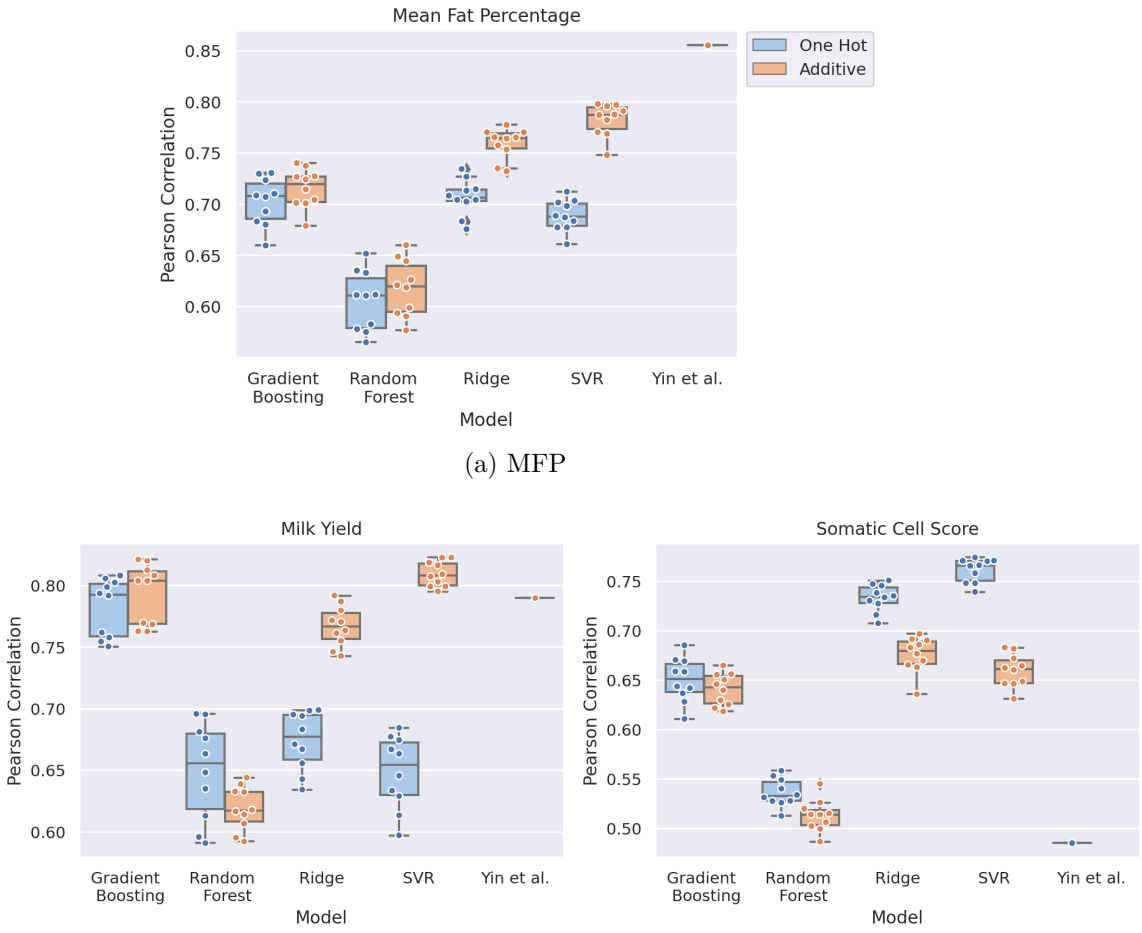


Figure 4.2: Predictive correlation by model (SVR for additive encoding exceeded the maximum computation time) and input encoding for Jersey Sire Conception Rate. Best results from (Rezende et al., 2019) are included for comparison.

4.4.2 Holstein

In a recent work (Yin et al., 2020), a new method for genomic prediction based on bayesian mixed linear models called KAML, was proposed, reporting best performance on traits with simpler genetic architectures. Using *classical* models we could not replicate this result, and predictive correlations did not show a clear association with trait complexity. As can be seen in Figure 4.3, the variation between splits was greatly reduced. This is probably due to higher sample sizes at training and testing, leading to greater stability regardless of the method used.

In contrast to the results obtained for the Jersey dataset, most models do not outperform the Bayesian Mixed Linear Model presented in (Yin et al., 2020). This may reflect the fact that more research has been done on the Holstein dataset than on the Jersey Dataset, which is a more lenient benchmark. The model with highest Correlation Coefficient in the test set was consistently SVM. This model outperforms the literature on MY and SCS, but does not match the high predictive correlations for MFP.



(a) MFP

(b) MY

(c) SCS

Figure 4.3: Predictive correlation by model and input encoding, for each trait. Best results from (Yin et al., 2020) are included for comparison.

As shown in Figure 4.3, in all traits, models trained on additively encoded data outperform their counterparts trained on One Hot encoded data. This result may be explained by the downsides of OHE mentioned in Section 4.4.1 (i.e: the loss of the one to one correspondence between biological markers and input features and the doubling of the input's dimensionality). Interestingly, One Hot Encoding appears to be less detrimental in Ensemble Methods (RF and GBM) than in Ridge and SVM. This may suggest that, in this problem, RF and GBM are better suited for categorical data, whereas the other regression methods are more adequate for ordinal data.

4.4.3 Wheat yield

The predictive correlations obtained were similar to those presented in (Crossa et al., 2014). Nonetheless, by means of extensive hyperparameter tuning, SVM

and GBM outperformed the results from (Crossa et al., 2014) in three out of four environments. Furthermore, Ridge regression exhibits lower predictive correlations than Ensemble Methods and SVM in all environments. As explained in Section 4, minimizing MSE with an L2 penalization corresponds to a MAP estimator with a Gaussian prior on the weights. Thus, Ridge's regression low predictive accuracy suggests that a Gaussian prior over the weights may not be appropriate for the Wheat dataset.

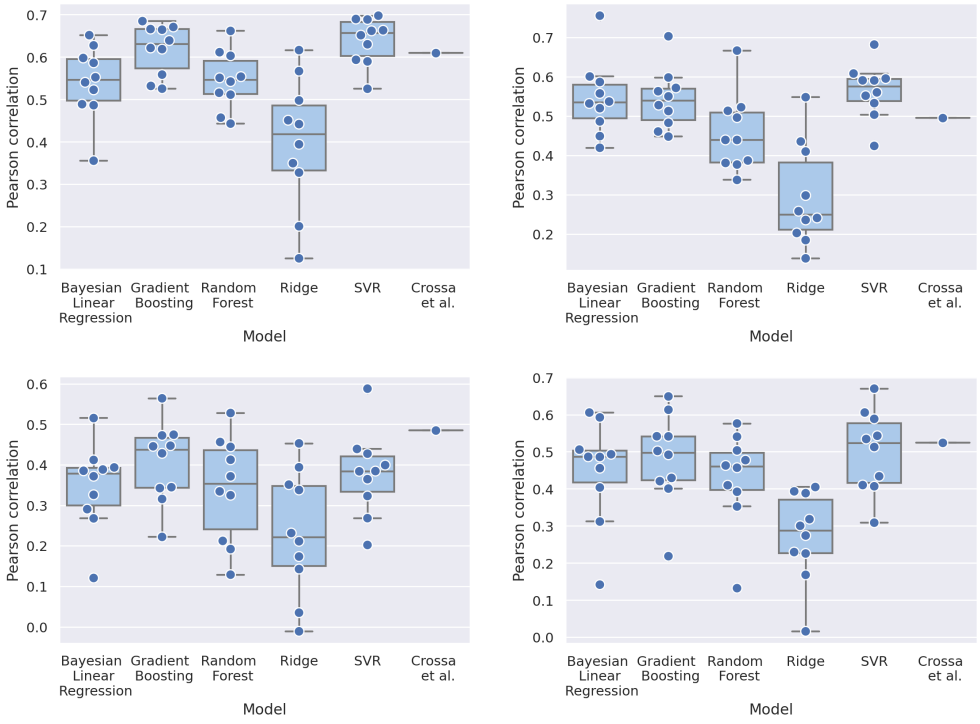


Figure 4.4: Predictive correlation by model and input encoding, for each environment. Best results from (Crossa et al., 2014) are included for comparison.

4.4.4 Yeast Growth

Similarly to the Jersey dataset, in Yeast, GBM exhibited higher predictive correlations than the rest of the models. Bayesian linear regressions performed comparably.

As shown in Table 4.1, in all four environments, the best classical model outperforms the results from (Grinberg et al., 2019). This illustrates the impact that exhaustive hyperparameter searches can have on model performance.

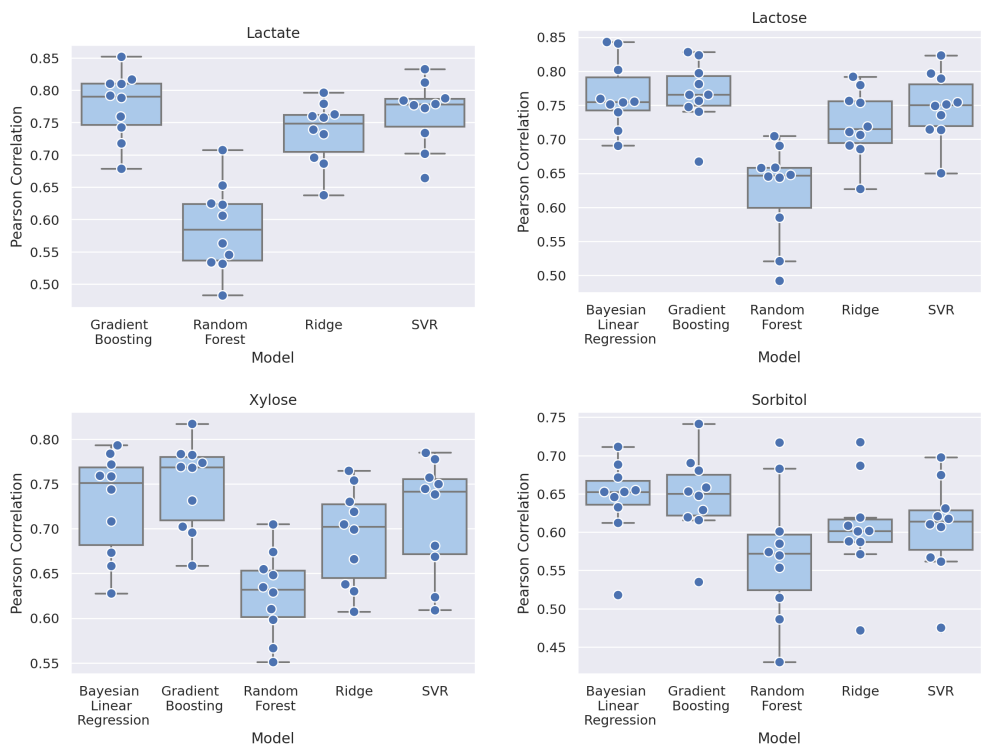


Figure 4.5: Predictive correlation by model and input encoding, for each environment. Best results from (Crossa et al., 2014) are included for comparison.

Yeast env.	Best from Grinberg et al. 2019	Best Classical Model (GBM)
Lactate	0.568	0.830
Lactose	0.582	0.860
Xylose	0.516	0.814
Sorbitol	0.424	0.681

Table 4.1: Best Coefficients of determination (R^2) obtained in classical models in Yeast and best results from Grinberg et al. (2019).

A description of the hyperparameters obtained with the randomized grid search is shown in Apendix G.

4.4.5 Summary

- Model performance showed high variance with respect to train/test splits.
- Hyperparameter tuning alone enabled surpassing the state of the art in Jersey, Yeast and Wheat dataset. Holstein proved to be more challenging.
- Additive encoding outperformed OHE in all datasets, RF and GBM were the least affected.

Chapter 5

Noise robustness experiments

5.1 Motivation

Although sequencing techniques are rapidly improving, phenotype prediction problems are affected by a great deal of noise (Grinberg et al., 2018). To evaluate the impact this might be having in our models, we propose a noise contamination experiment.

The experiment consists in purposefully contaminating the data with different types of noise and subsequently evaluating the performance of methods described in 4. The three noise sources considered were phenotypic noise, missing markers and missing samples, which are some of the most common problems when dealing with genomic data. Through these tests we aim to measure each of the model's robustness towards different types of noise in the data.

5.2 Experiments description

Phenotypic noise refers to the noise present in the measured phenotypic trait and has two main components: environmental conditions and the measuring process itself. As previously mentioned, the phenotypic value of an individual depends not only on its genotype but also on the environment that surrounds it (i.e. weather, water availability and soil conditions for crops). In addition, the measuring process introduces an additional noise source which is not always negligible. To evaluate the impact of such noise in our datasets, we randomly selected 10, 20, 30, 50, 60, 70, 80 and 90 percent of samples of the phenotype data and contaminated them by adding or subtracting (with equal probability) two times the standard deviation of the whole dataset.

The second type of noise considered (missing markers) consists of randomly selecting a portion of the genotype's markers and deleting them (same markers for all individuals). Sequencing techniques are as potent as ever, enabling ever growing sampling rates. Although denser sampling may seem beneficial at first glance, linkage disequilibrium can make many markers redundant. Moreover, the increase in the input's dimensionality should be matched with more sequenced

individuals, which is not always the case. On the other hand, the markers used for prediction could be insufficient to achieve an accurate representation of the genome. This experiment aims to evaluate the extent to which denser sampling is beneficial to the model's performance. The proportion of deleted markers ranged from 10 to 99 percent.

Finally, the missing samples evaluation consists of randomly selecting and deleting a portion of samples (which means they are not used for training nor testing). This study evaluates the extent to which the amount of samples collected limits performance. Furthermore, it may give us a glimpse into how much predictions could improve if more individuals were sequenced. In this case, the deletion ratio ranged from 10 to 80 percent.

5.3 Methodology

The methodology for a single experiment is summarized in Algorithm 1. An experiment is comprised of a model, a dataset, an environment or trait, a noise type and a fixed seed.

Algorithm 1: Noise robustness experiment

Result: Test Pearson correlation for each contamination ratio
Initialize result buffer
for *ratio in ratios* **do**
    Contaminate genotype/phenotype matrix
    Split into train, validation, and test splits
    Tune model hyperparameters using train and validation splits
    Re-train best hyperparameters in train and validation splits
    Predict on test split and calculate Pearson correlation
    Store metric in result buffer
end

We select up to three different environments or traits per dataset. Each dataset is split into 5 different random splits with 60%, 20%, and 20% train, validation, and test samples respectively. We begin by calculating the base (i.e. 0% markers/samples contaminated) performance for each split. We then proceed to run Algorithm 1 on these same splits, which produces the Pearson correlation for each contamination ratio. These values are then normalized to represent the relative performance w.r.t. the uncontaminated case. Lastly, we average the relative performance over all splits and environments/traits.

The experiment is slightly different for the missing samples experiment. In this case, the contamination and split order is reversed, and the samples are dropped only from the train/validation set. Otherwise, the test set would change and get progressively smaller with each contamination ratio.

All random effects (dataset splitting and contamination) are seeded to ensure a fair comparison between the different models.

These experiments were not performed on the Jersey bulls dataset due to the dimensionality of the signal, which made running several contamination ratios per split computationally prohibitive for most models. Similarly, Random Forest experiments on the Holstein dataset could not be conducted due to computational restrictions.

5.4 Results

Figures 5.1, 5.3 and 5.2 show the results obtained for the missing markers experiment on the yeast, wheat and Holstein datasets. The thicker line represents the mean relative performance drop across all splits and environments/traits, while the lighter fill represents the standard deviation.

Overall, the three datasets exhibit a similar behaviour: performance does not significantly drop until a high amount of markers are removed. Once this value is surpassed, performance rapidly declines. The amount of markers that can be removed before the model's performance starts decreasing varies among datasets. In yeast, the threshold is approximately 80 percent of the total markers, with RF losing performance sooner (at around 60 percent). Wheat is the most affected dataset, with Ridge and RF showing a monotonic reduction in performance. GBM and SVR are more robust towards marker elimination in this dataset, showing signs of performance degradation at 50 and 60 percent respectively. Lastly, the Holstein dataset shows similar results for its three models (GBM, Ridge and SVR), with performance degrading from 50 percent onward.

The fact that large portions of the signal can be removed before performance significantly drops can be attributed to high linkage disequilibrium between SNPs. In other words, densely sequenced genomes exhibit highly correlated features, some of which may be redundant for prediction purposes. This is consistent with the fact that the Wheat dataset, which exhibits the lowest degree of linkage disequilibrium (see Appendix B.2), was the most affected by marker deletion.

The results from the phenotypic noise and missing samples experiments can be found in appendix F.

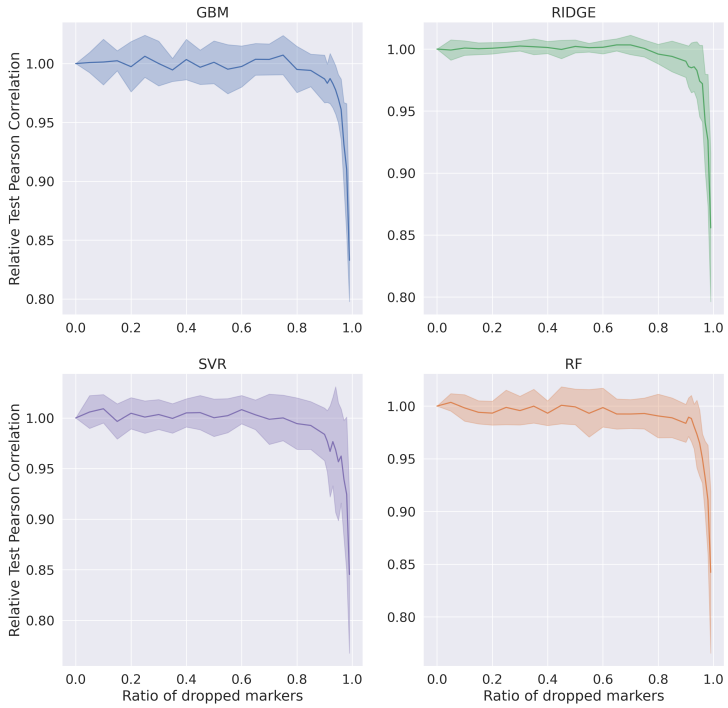


Figure 5.1: Performance of different methods for the Yeast dataset under varying amounts of dropped markers.

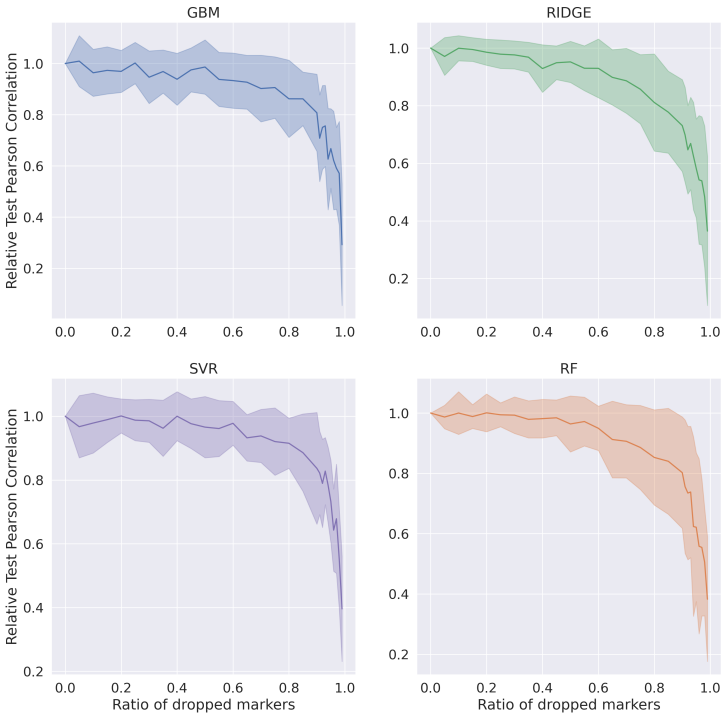


Figure 5.2: Performance of different methods in the Wheat dataset under varying amounts of dropped markers.

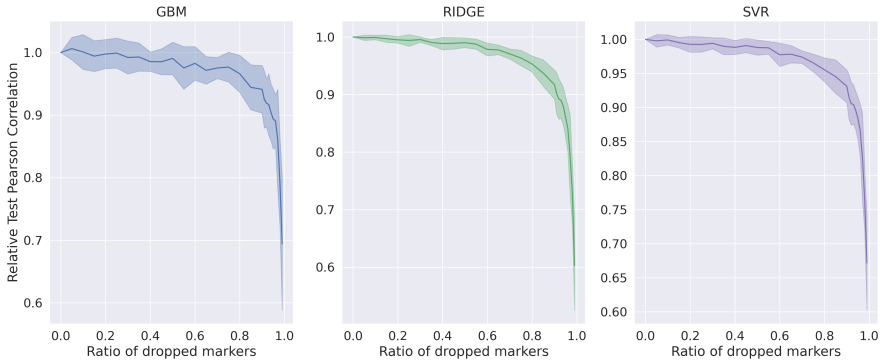


Figure 5.3: Performance of different methods in the Holstein dataset under varying amounts of dropped markers.

This page intentionally left blank.

Chapter 6

Convolutional Neural Networks

6.1 Introduction

Deep Learning has unquestionably established itself as the leading paradigm to deal with unstructured data. The recent rise of architectures such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), Graph Neural Networks (GNNs), and Transformers resulted in an unprecedented breakthrough for different tasks in computer vision, natural language processing, speech and reinforcement learning. Both gradient-descent based optimization and hierarchical representation learning play a significant role in the superior performance of these algorithms.

We naturally wonder if we can replicate these results in the context of genome-wide phenotype prediction. Most agricultural phenotypes have a multifactorial inheritance, with multiple and complex relationships across genes and between genes and their environment (Abdollahi-Arpanahi et al., 2020). Consequently, model free approaches such as the ones mentioned above might be better suited to tackle these intrinsic challenges to genomic data.

We will begin exploring Deep Learning's applicability by studying CNNs. A CNN is a particular kind of network which exploits spatially correlated data. These networks have proven to capture semantic information from unstructured data such as images and audio (Hershey et al., 2016; Krizhevsky, Sutskever, & Hinton, 2012). Although their generalization power is yet to be fully understood (C. Zhang, Bengio, Hardt, Recht, & Vinyals, 2016), some factors contributing to their success have been recognized. Firstly, by learning convolutional kernels via Stochastic Gradient Descent (SGD), CNNs enable automatic feature extraction[1]. Moreover, CNNs inherit inductive biases from their convolutional backbone, such as translational equivariance. The impact and adequacy of these inductive biases in the context of genome-enabled phenotype prediction is discussed in Chapter 6.5.

Moreover, CNNs drastically reduce the number of parameters in comparison to fully-connected networks. Batch Normalization and Dropout techniques, first introduced in CNNs, have alleviated the difficulties of training deep, over

[1] Automatic feature extraction is not restricted to CNNs.

parametrized models. Lastly, the success of CNNs -and deep learning in general- can not be decoupled from the creation of cheaper and faster hardware nor the dramatic increase in volume of labelled image data.

In our case, the motivation behind using a CNN stems from the Linkage Disequilibrium and Cross Marker Interaction phenomena (Abdollahi-Arpanahi et al., 2020). As mentioned in Chapter 1.1, linkage disequilibrium causes markers to be locally correlated, much like pixels in an image. Ideally, a CNN should exploit these underlying local correlations better than a traditional multi-layer perceptron model, while requiring a significantly lower number of parameters.

Still, deep learning techniques require high amounts of data compared to the algorithms described in Chapter 4. To make matters worse, the amount of data needed generally scales with the dimension of the input signal, which in our case is considerably high. Therefore, deep learning approaches will only be tested in the yeast and German Holstein datasets, which provide a suitable trade-off between dimensionality and number of samples 3.1.

6.2 Building blocks

Although CNNs are defined by the presence of convolutional layers, other distinct layers are often incorporated into their architectures. In this Section we provide a brief overview of these layers, namely Convolutional, Pooling, Dropout, Batch Normalization, and several point-wise non-linearities.

6.2.1 Fully Connected Layers

Fully Connected Layers (FCLs) are named that way because they connect every input feature to every output feature. Mathematically, a FCL implements the transform

$$\mathbf{x}^{\text{affine}} = \mathbf{W}^{\mathrm{T}}\mathbf{x} + \mathbf{b}$$

where $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{W} \in \mathbb{R}^{N \times M}$, $\mathbf{b} \in \mathbb{R}^M$, and $\mathbf{y} \in \mathbb{R}^M$. An example of this transformation for $N = 6$ and $M = 4$ is shown in Figure 6.1.

This affine transformation is often followed by a point-wise non-linearity function, such that $\mathbf{y} = \sigma(\mathbf{x}^{\text{affine}})$. By stacking multiple FCLs and non-linearities in series, the network is able to learn complex, non-linear relationships between input and output. Neural Networks with this architecture are often referred to as Fully Connected Networks (FCNs) or Multi-layer perceptrons (MLPs). In this work, we will use these two terms interchangeably.

In the context of CNNs, FCNs are often included after vectorizing (i.e. flattening) the last Convolutional Layer's feature maps, which enables learning of non-linear interactions between the CNN's output features. The last layer of the FCN is responsible for the final prediction of the target variable, which in our case is the individual's phenotype(s).

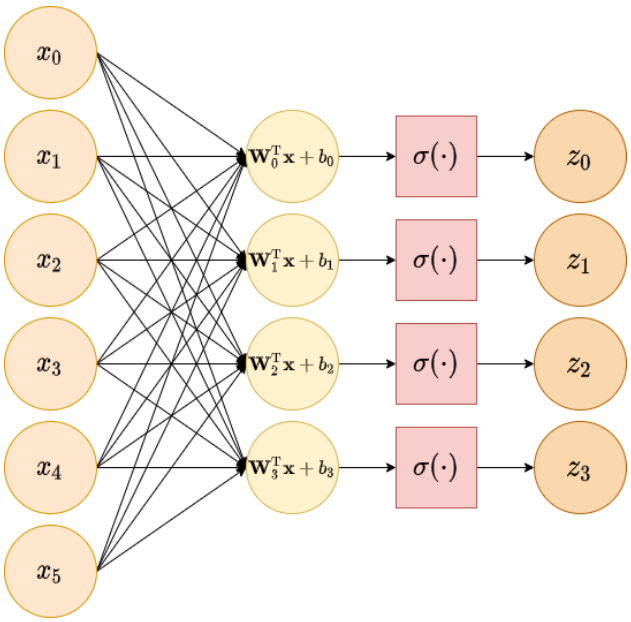


Figure 6.1: Example of a FCL. $\mathbf{W_i}$ and b_i refer to the i-th row of the layer's weight matrix and the i-th component of the bias vector, respectively. The circles represent input, intermediate, and output features.

6.2.2 Convolutional layers

Convolutional layers are the core components of any CNN architecture. They consist of a set of learnable filters (i.e. a filter bank), such that each individual filter is convolved with the input feature maps during the forward pass. The discrete convolution operation between a one-dimensional[2] signal x and a filter w is defined as

$$(x \circledast w)[n] = \sum_{m=-\infty}^{\infty} x[m]w[n-m]$$

which is equivalent to sliding the filter across the input signal while computing the dot product[3] between its entries and input's. Since the filter w is optimized via SGD, the convolutional layer learns filters that detect specific types of spatial patterns across the input signal. By stacking multiple filters, these layers are able to detect many different types of patterns in the input. Thus, each filter introduces an independent channel in the output feature map. This can be visualized in Figure 6.2.

Moreover, convolutional layers can be stacked in series, with each subsequent layer convolving the output feature map of the last. This leads to richer feature maps, which can contain relevant semantic information for the downstream prediction task.

[2]for two-dimensional convolutions, the filter slides across the height and width of the input.

[3]in Deep Learning, a learnable bias term is usually added to the dot product.

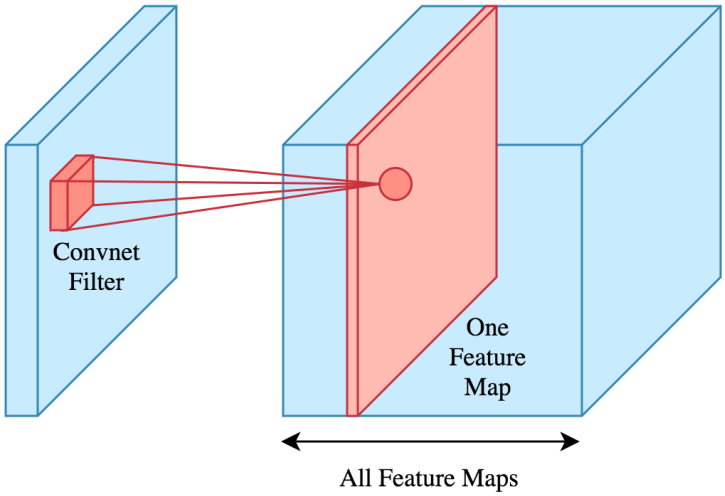


Figure 6.2: Example of the forward pass of a single two-dimensional convolutional filter (in red). The filter slides across the width and height of the image, usually in a left-to-right, top-to-bottom manner. Each filter produces exactly one output feature map (i.e. channel).

Besides, CNNs introduce other beneficial properties such as weight sharing. When dealing with high-dimensional signals such as images or audio, a traditional linear (fully connected) layer would introduce a considerably higher number of parameters than a convolutional one. In addition, linear layers do not take the signal's spatial structure into account. Convolutional layers solve this two issues via their sliding filters, which contain considerably less parameters (which are shared across the input) and enforce local connectivity between subsequent layers.

Lastly, convolutional layers introduce translational *equivariance*, a useful inductive bias in fields such as image and audio recognition. This bias stems from the sliding dot product in the forward pass, which implies that if the input signal is translated, the output feature map is translated by the same amount. More recently, Group Equivariant Convolutional Layers have been proposed, which enforce rotational and reflectional equivariance (Cohen & Welling, 2016).

Whether the inductive biases (translational equivariance, local connectivity) listed above are suitable for the genome-wide phenotype prediction will be discussed in Section 6.5.

6.2.3 Pooling layers

Pooling layers implement a form of non-linear feature map downsampling. They where introduced by LeCun et al. (1989) in the context of hand-written digit recognition, in what is known as the first successful application of a CNN in a visual recognition task. Pooling layers are commonly inserted in between convolutional layers to reduce the feature maps' spatial size (which is typically compensated by an increase in the number of channels).

Pooling layers can be modelled as a sliding filter of stride $s > 1$, where s

represents the downsampling factor. The filter slides across the input feature maps, aggregating the activations inside its receptive field via an aggregating function. This process is applied in a channel-wise manner, downsampling each channel individually (i.e. the number of channels is unchanged). This process can be visualized in Figure 6.3, where each color represents a different position of the filter.

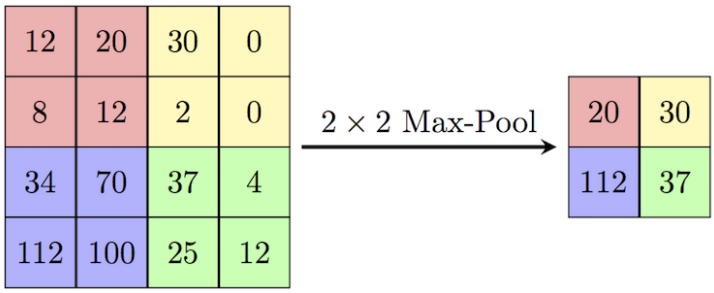


Figure 6.3: Example of a Max-pooling layer with stride 2 and kernel size 2×2. The output feature map has $1/4$th of the original input's features.

The motivation behind pooling layers arises from both performance and computational reasons. The pooling operation introduces a certain degree of translational *invariance*, which is beneficial in certain tasks such as classification. In addition, the feature map downsampling reduces the amount of parameters in the network, which reduces memory footprint and increases both training and inference speed, and naturally combats overfitting.

The most popular form of pooling is max-pooling, in which the aggregating function consists of taking the maximum activation inside the receptive field. However, other forms of pooling exist, such as average or ℓ_2-norm pooling.

It is worth noting that pooling layers are being often replaced by strided convolutions (convolutional layers whose stride is greater than one) (He, Zhang, Ren, & Sun, 2015a; Xie, Girshick, Dollár, Tu, & He, 2017; Zagoruyko & Komodakis, 2016).

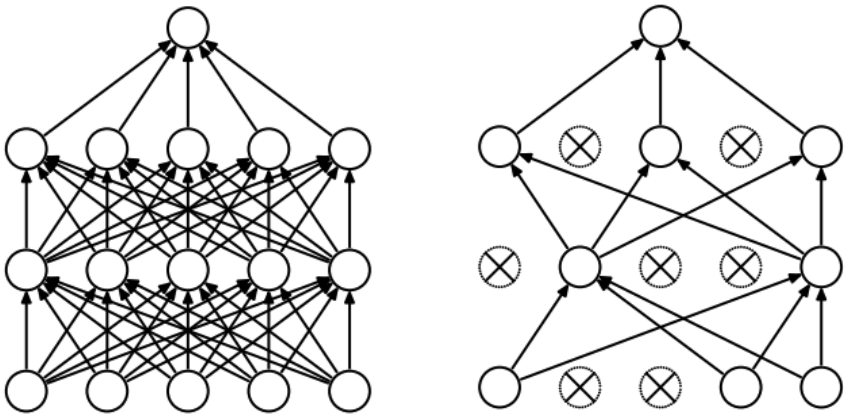
6.2.4 Dropout

Dropout layers consist of a regularization technique in which input activations are dropped (i.e. set to zero) with probability p (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). The gradients w.r.t. dropped units are zero, which implies that parameters associated with these units are not updated. This amounts to sampling a different sub-network in each train iteration, with higher values of p leading to smaller sub-networks and thus stronger regularization. This network sub-sampling prevents units from co-adapting too strongly, which reduces the model's capacity and leads to more robust features which generalize better.

Although the combination of sub-networks might resemble an ensemble method, this is not the case for Dropout neural networks, since parameters are still trained

jointly across different training epochs. For a deeper discussion on dropout see Mianjy, Arora, and Vidal (2018) and Gal and Ghahramani (2016).

The effects of activation dropping are visualized in Figure 6.4.



(a) Without dropout (b) With dropout

Figure 6.4: Example of a MLP with and without Dropout layers. The crossed units correspond to dropped activations. Image extracted from (Srivastava et al., 2014).

6.2.5 Batch Normalization

Batch Normalization (BN) layers were originally proposed in (Ioffe & Szegedy, 2015) as a means to make deep networks faster and more stable to train by addressing the internal covariate shift phenomenon. During training, the network's parameters are updated iteratively, which means the distribution to the intermediate layer's input changes with each epoch. The authors of the BN paper theorize that this internal distribution shift makes it slower for the parameters to converge to the local minima.

BN layers solve this issue by normalizing input features using the batch's statistics. Thus, if the input to the BN layer is $\mathbf{x} = (x_0, x_1, ..., x_{d-1})$, its output $\hat{\mathbf{x}} = (\hat{x}_0, \hat{x}_1, ..., \hat{x}_{d-1})$ is given by

$$\hat{x}_i = \frac{x_i - \mu_{B_i}}{\sqrt{\sigma^2_{B_i} + \epsilon}}$$

where

$$\boldsymbol{\mu}_B = \frac{1}{b}\sum_{n=0}^{b-1} \mathbf{x}_n$$

$$\boldsymbol{\sigma}^2_B = \frac{1}{b}\sum_{n=0}^{b-1} (\mathbf{x}_n - \boldsymbol{\mu}_B)^2$$

with b being the size of the batch. The constant ϵ is small positive number added for numerical stability purposes. In practice, a moving average of the batch's

statistics is used instead of considering the present batch's statistics only. In addition, feature-wise learnable shift and scale parameters are implemented to preserve the model's expressiveness, such that

$$y_i = \gamma_i \hat{x}_i + \beta_i, \qquad \beta_i, \gamma_i \in \mathbb{R}$$

Another benefit of BN layers is that calculating statistics over the batch introduces a noise component on the gradients, which acts as an additional source of regularization.

Although BN layers have been widely adopted by the Deep Learning community, their exact working mechanism is still matter of debate. (Santurkar, Tsipras, Ilyas, & Madry, 2019) show that the distributional stability of intermediate inputs has little impact on BN's effectiveness. Instead, they propose that BN smoothens the loss landscape, resulting in a more stable and predictable behavior for the gradients.

6.2.6 Non-linearities

Pointwise non-linear activation functions are applied to each layer's output feature map and in between FCLs, increasing model capacity and allowing the network to learn complex non-linear relationships. The choice of such functions has a great impact on the optimization landscape, training speed, and model performance. We outline some of the most popular ones below, which were used in the proposed architectures, and are shown in Figure 6.5.

Tanh

The hyperbolic tangent $tanh : \mathbb{R} \to [-1, 1]$ is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

It has the desirable property of having a zero centered output (Glorot & Bengio, 2010a), and can be interpreted as a re-scaled and centered version of the Sigmoid function:

$$\tanh(x) = 2\sigma(2x) - 1,$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the Sigmoid function.

One problem that arises is that if the input to the Tanh layer has large values (in absolute terms), its gradients are near zero. In other words, the output saturates and training may slow down or even stop altogether. Weight initialization should thus avoid saturation in order to allow faster training, especially in very Deep Networks (Glorot & Bengio, 2010a).

ReLU

First proposed by Fukushima (1969) and more recently popularized by Krizhevsky et al. (2012), the ReLU activation function $relu : \mathbb{R} \to \mathbb{R}^+$ is defined as

$$\text{ReLU}(x) = \max(0, x)$$

Apart from being computationally cheaper, due to its linear, non-saturating region, ReLUs have been found to accelerate convergence when compared to the Tanh, especially in very Deep Networks (Krizhevsky et al., 2012). The saturating negative region causes sparser activations, which can lead to better solutions (Glorot, Bordes, & Bengio, 2011; Nair & Hinton, 2010). However, it can also make training difficult, particularly without a proper initialization (He, Zhang, Ren, & Sun, 2015b; Lu, Shin, Su, & Karniadakis, 2019).

Leaky ReLU

This activation function introduces a small slope (Maas, Hannun, & Ng, 2013) in the negative region of the ReLU activation, so as to prevent 'dead' neurons, that is Neurons that don't fire. Moreover, the slope improves the gradient flow across the network when compared to the standard ReLU, since dead neurons now have non-zero gradients. This effect is particularly useful in very Deep Networks, so as to counter the vanishing gradients problem (Xu, Wang, Chen, & Li, 2015).

The Leaky ReLU is defined as

$$\text{LeakyReLU}(x) = \max(\alpha x, x) \ , \ 0 < \alpha < 1,$$

where α is a hyperparameter usually set to 0.01. In *Parametric* ReLU neurons (He et al., 2015b) α is learnt jointly with the model. Leaky ReLUs have also been found to outperform ReLUs in some tasks (Xu et al., 2015).

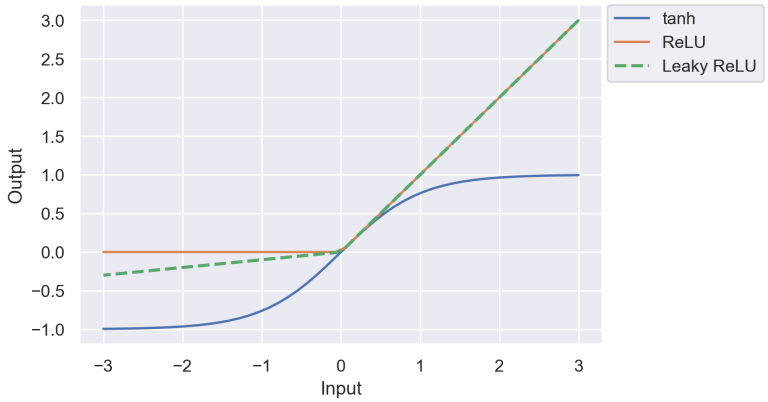


Figure 6.5: Non-Linear activation functions. Leaky ReLU has $\alpha = 0.1$.

6.3 One dimensional

6.3.1 Related work

Although recent genomic prediction works have incorporated deep learning models, deep learning remains rather unexplored in this context. Moreover, their 'shallow' counterparts remain unbeaten in both performance and interpretability. W. Ma

et al. (2018) implement a dual-branch CNN to predict five different phenotypes in a soybean dataset. They achieve consistent but marginal gains in performance (0.01 Pearson correlation improvement) compared to regularized linear regressions. Similarly, Liu et al. (2019) utilize a CNN to predict eight phenotypes in a wheat dataset. The results show an improvement of 0.005 in Pearson correlation with relation to the rrBLUP (Endelman, 2011) and GBLUP (Clark & van der Werf, 2013) methods. In a more recent work (Abdollahi-Arpanahi et al., 2020), compared the performance of multi-layer perceptrons (MLPs) and CNNs versus ensemble and parametric methods in a Holstein bull dataset. Both deep learning approaches achieved the lowest performance, with the CNN having a relative gain of 10% over the MLP in terms of Pearson correlation.

6.3.2 Proposed architectures

AlexNet-like CNN

The network's most basic architecture consists of two subsequent convolutional blocks followed by two fully connected layers. Convolutional blocks are composed of a one-dimensional convolutional layer with a leaky Rectified Linear Unit (ReLU) activation, followed by a batch normalization and max-pooling layer. In addition, a dropout layer was added after each convolutional block to combat overfitting.

This architecture is heavily inspired by the AlexNet CNN (Krizhevsky et al., 2012), which established Deep Learning's first major breakthroughs in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Moreover, AlexNet marked the beginning of Deep Learning's dominance over traditional computer vision approaches. By adopting a similar architecture, we hope to replicate the AlexNet phenomenon in the context of genome-wide phenotype prediction.

The main additions to the AlexNet-like architecture are the Dropout and BN layers. Figure 6.6 shows a detailed diagram of the model, along with its hyperparameter settings.

Residual CNN

Inspired by the success of ResNet-like architectures in computer vision (He et al., 2015a), we experimented with residual blocks in between the convolutional and fully connected sections of the network. Residual connections were initially proposed to counteract the vanishing gradient problem that arises in very deep networks and to facilitate learning identity mappings He et al. (2015a).

Residual connections are called that way because they learn the *residual* as a transformation of the input. Figure 6.7 illustrates the basic structure of a residual block. From the image it follows that

$$\begin{aligned} \mathcal{R}(x) &= \text{output} - \text{input} \\ &= \mathcal{F}(\mathbf{x}) + \mathbf{x} - \mathbf{x} \\ &= \mathcal{F}(\mathbf{x}) \end{aligned} \tag{6.1}$$

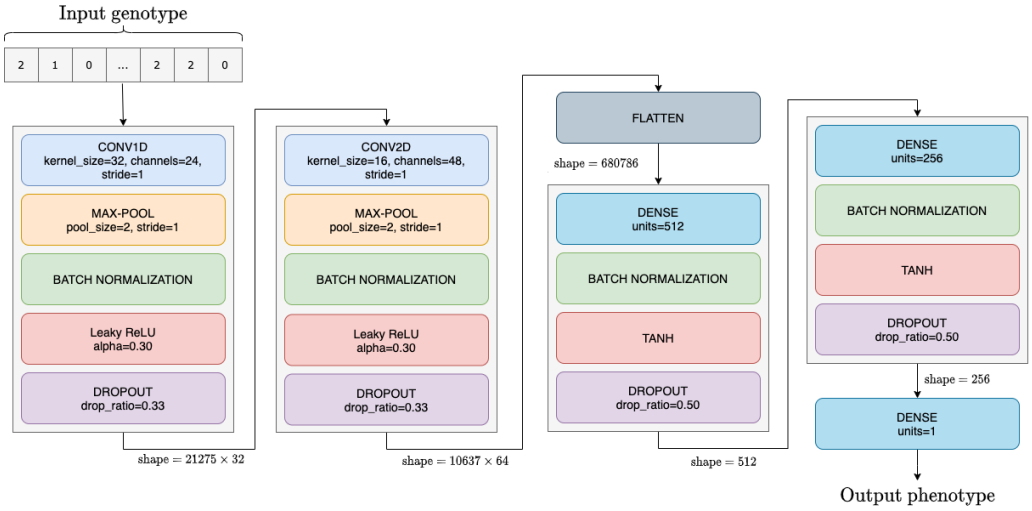


Figure 6.6: AlexNet-like CNN architecture.

Thus, the residual block's non-linear layers are learning the residual between its input and the optimal output. During training, skip connections improve the flow of gradients across the network. This helps combat vanishing gradients in very deep networks, making earlier layers learn faster. In addition, if the optimal mapping of a layer could be expressed as an identity, the solver can simply drive the weights of the non-linear layers to zero.

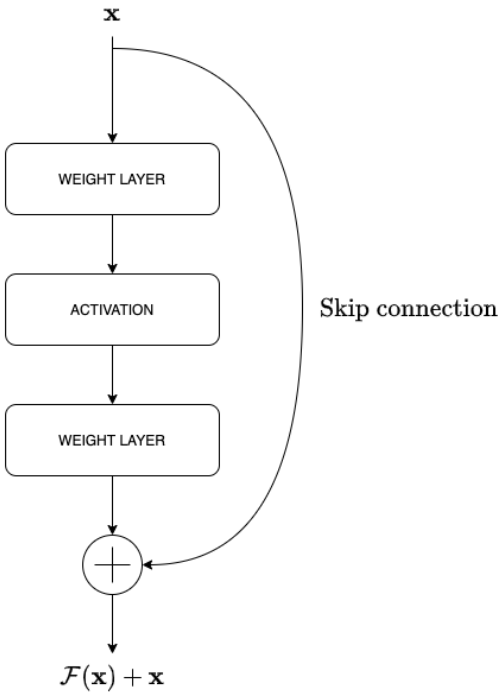
(He et al., 2015a) argue that even though identity mappings are rarely optimal, the reformulation still helps the network's learning and generalization. If the optimal transformation is closer to an identity mapping than to a zero mapping, it should be easier for the network to learn the perturbations around the identity than to learn the optimal transformation from scratch. To support this claim, the authors run several experiments, showing that the residual functions in general have small responses. These results suggest that the identity preconditioning imposed by residual blocks eases the learning task.

In practice, the input and output of our residual blocks have different channels. Thus, we also train a linear projection to map the input to the appropriate output dimension[4].

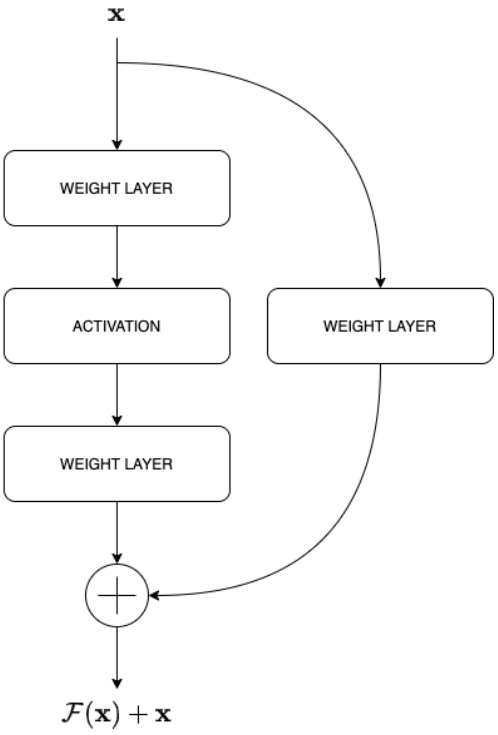
To the best of our knowledge, this is the first application of a Residual CNN in this particular task. Still, similar architectures have been applied to other omics[5] problems. D. Wang et al. (2017) use a residual CNN architecture to predict backbone torsion angles from amino acid sequences. Fang, Shang, and Xu (2019) employ a residual CNN with an attention mechanism to predict phosphorylation sites from protein fragments.

[4]The linear projection is then fed to a leaky ReLU activation, which is non-linear.

[5]Disciplines in biology whose names end in the suffix -omics, such as genomics, proteomics, metabolomics



(a) Residual block



(b) Residual block with linear projection

Figure 6.7: Residual blocks.

To prevent over regularization due to the residual connection, the Dropout layers in between convolutional layers were removed. Figure 6.8 shows the Residual CNNs architecture, as well as its hyperparameter settings.

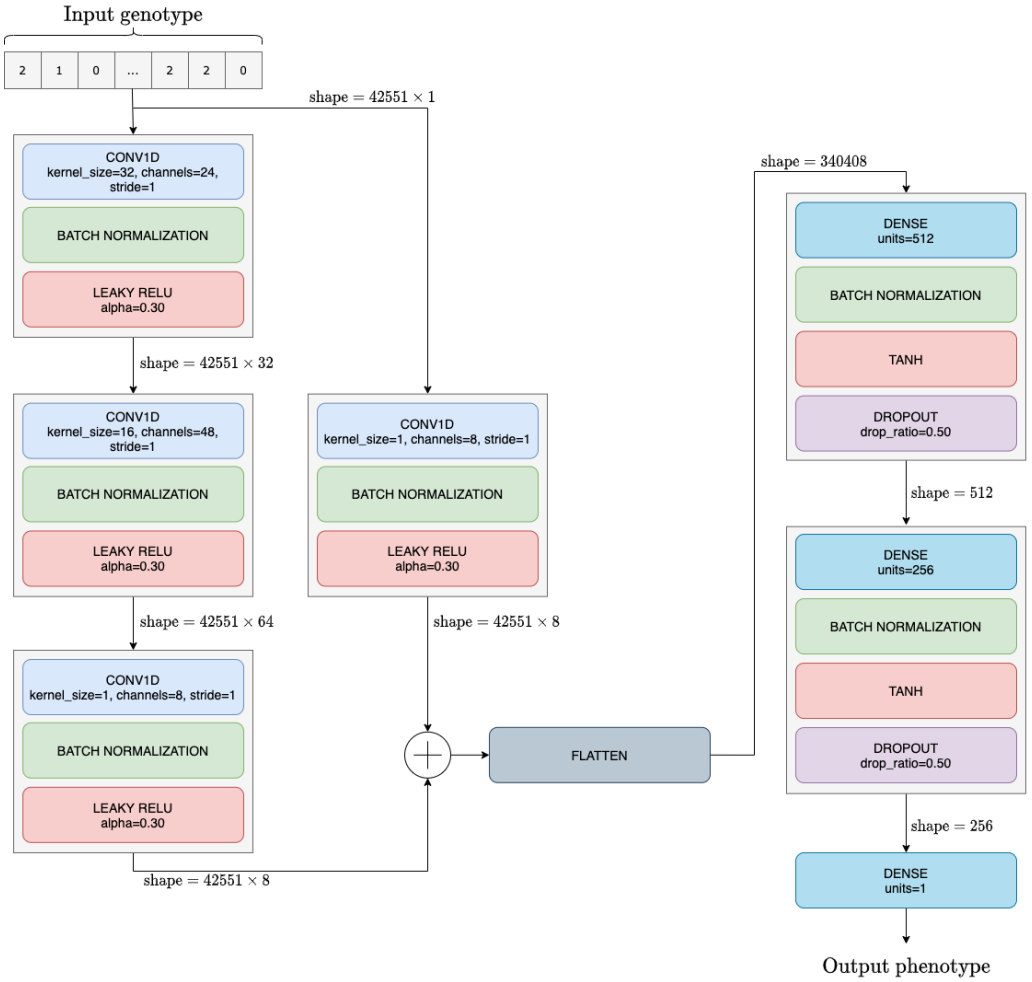


Figure 6.8: Residual CNN architecture.

6.3.3 Hyperparameters

Hyperparameters such as kernel size and number of filters have been studied exhaustively in fields such as computer vision. To the best of our knowledge, there's no previous research or heuristics on hyperparameter tuning in this task. The kernel size in particular controls how many neighboring features are aggregated by each filter. This parameter should reflect the degree of linkage disequilibrium in the input genotype. However, this information is hard to measure accurately in practice. To make matters worse, local correlations are not homogenoeous along

the genome, as the length of haplotype blocks may vary. Although unsupervised learning techniques such as clustering can give some insight into the genotype's structure (see Section B.3), results vary greatly between different techniques and parameter settings. Consequently, the problem of choosing appropriate hyperparameters is tackled via exhaustive randomized searches.

The only design rules we adopted are: (i) if the feature map size is halved, the number of filters is doubled to maintain the time complexity per layer; and (ii) kernel sizes and fully-connected units are halved after consecutive each layer (i.e. if the first convolutional block has a kernel size of 256, the second and third blocks will have 128 and 64, respectively). The complete list of hyperparameters searched along with their values can be found in Table 6.1.

Hyperparameter	Search type	Values
Number of CNN layers	Choice	{2, 3, 4}
Number of CNN filters (1st layer)	Choice	{8, 16, 24, 32}
CNN kernel size (1st layer)	Choice	{32, 64, 128, 256, 512}
Number of FC layers	Choice	{2, 3, 4}
Number of FC units (1st layer)	Choice	{128, 256, 512}
FC activation	Choice	{tanh, ReLU, LeakyReLU, ELU}
Dropout rate	Uniform	min = 0.25 max = 0.50

Table 6.1: CNN hyperparameter search space

6.3.4 Multi-task learning

As mentioned previously, the low number of observations is one of the main limiting factors in genomic prediction. In light of this issue, we propose to frame the problem of complex phenotype prediction as a multi-task one (Ando & Zhang, 2005; Caruana, 1997; Evgeniou & Pontil, 2004).

Incorporating information from other environments naturally combats overfitting. Having different outputs for the same input (one for each environment or trait) may force the model to extract features with more robustness towards environmental or trait-specific noise. This robustness can lead to an increase in performance for individual environments.

Different traits or environments tend to be weakly correlated. Thus, we experiment with including all of them versus including a small group of highly correlated ones. As shown in Chapter 3.1, the Holstein dataset exhibits only two correlated traits (Milk Yield and Somatic Cell Score) with a Pearson correlation of 0.60. In

the Yeast dataset, the chosen subset of environments consists of four sugar-related environments which exhibit a medium to high pairwise correlation (0.48 to 0.80).

6.3.5 Methodology

We distinguish four different CNN architectures: the base AlexNet-like CNN and the residual CNN, with their corresponding single and multi-trait variants. The hyperparameters of the Residual CNN and the AlexNet-like CNN were determined independently via a randomized grid search as described in the previous Section, and are maintained across all different experiments.

Each architecture is evaluated with the base marker order (referred to as unshuffled) and the shuffled markers, for a total of eight distinct CNN models. In addition, we compare the CNNs to a single-trait, 5-layer MLP described in Table 6.2. The MLP is completely agnostic to marker position, which means it cannot exploit the genome's local structure to the extent the CNN can. Thus, the CNNs together with the MLP comprise nine different *experiment types*.

Each experiment type is ran on 20 different splits with 70% of the data reserved for training, 10% reserved for validation, and the remaining 20% for test. The splits are seeded so that all experiments run on the same 20 splits. All models are trained with the Adam optimizer, with a base learning rate of 10^{-3} during 120 epochs. The learning rate is reduced by a factor of 10 in epochs 30 and 60. In addition, an early stopping callback is implemented to prevent overfitting. All weight matrices were intitialized via the Glorot Uniform initializer (Glorot & Bengio, 2010b), while biases were initialized to zero.

Maintaining the same splits and model configuration across experiments implies that the only source of variation is the specific changes each of them introduce. This allows us to extract more reliable conclusions about each variation's impact on the prediction task.

Layer	Units	Dropout	BatchNorm after dropout	Activation
0	1024	0.50	Yes	ReLU
1	512	0.35	Yes	ReLU
2	256	0.30	Yes	ReLU
3	256	0.30	Yes	ReLU
4	256	0.25	Yes	ReLU

Table 6.2: MLP parameters.

6.3.6 Ablation studies

To evaluate if the CNN architecture is exploiting the genotype's structure, we propose an ablation study in which we randomly shuffle the order of the markers. This random shuffling completely breaks any local structure that the genotype might

have. If the CNN's inductive biases were working as expected, its performance on this shuffled data should drop drastically.

In addition to the ablation study, we also train a 4-layer MLP with BN and Dropout layers. This model is completely agnostic to marker position and as such we expect it to achieve lower performance than its convolutional counterpart.

6.3.7 Model interpretation

Saliency Maps

Traditional models in genome-enabled prediction such as bayesian linear regressions can be easily interpreted, since each weight in the model is associated to one input feature. In these models, the relative importance of features can be assessed by comparing weight magnitudes. Similarly, Decision Tree (DT) based models provide several criteria to evaluate feature importance, such as the individual performance gain of DTs which use a specific feature to split at least one node. Another criteria for DT based models is the number of DTs that split a node using a specific feature.

However, CNNs can have multi-layer architectures and can be highly overparametrized (see Section 7.7.10), which makes the mapping from input to output more complex. This may enable the model to capture complex interactions between input features, which might come with the downside of leading to harder to interpret models.

Certain tools that attempt to gain insights into the inner workings of these *black box* models have been developed. Saliency maps are an example of such tools. Specifically, saliency maps indicate the magnitude of the gradient of the output with respect to each feature of an input sample:

$$s = \nabla_x(f(x))$$

where f is the model and x an input sample. Since a local first-order approximation of the model is: $f(x) \approx x\nabla_x(f(x))$, saliency maps can be viewed as the weights of a first order approximation of the model centered at point x.

Activation maps

Activation maps consist of representations of the activation functions at intermediate layers. Larger activation values (in absolute terms) have a stronger influence on the network's prediction. Thus, activation maps can help determine which features or patterns have the biggest impact on prediction.

6.4 Two-Dimensional

6.4.1 Motivation

As described in Section 6.1, convolutional models exploit local structure in signals. In the case of genomic sequences, this local structure may correspond to neighbor-

ing SNPs in linkage disequilibrium. However, as can be seen in the hierarchical clustering experiments B.2, related SNPs can also be found far away from each other. Thus simply sliding a convolutional kernel through the genomic sequence may prevent the model from capturing complex interactions between distant SNPs.

This suggests that it may be beneficial to find a representation of the genome in which similar SNPs are close and dissimilar SNPs are further apart. Such alternative representation of the genome could catalyze the extraction of structural information via convolutions. In this sense, an image-like representation seems like a coherent choice since it also enables the use of popular image processing tools and two-dimensional[6] CNNs.

In the following section, we describe the scheme used to transform genomic sequences into image-like samples. We then use these image datasets to train CNNs to predict Yeast growth and Holstein milk yield. Finally, we explore the limitations of this approach and propose an alternative definition of *similarity*, suitable for high dimensional vectors.

6.4.2 Genome to image

Sharma, Vans, Shigemizu, Boroevich, and Tsunoda (2019) propose a technique called *DeepInsight* (DI) for transforming a set of numerical sequences into a set of image-like matrices. To attest the generality and usefulness of the DI method, Sharma et al. apply it to three real datasets of different nature:

- RNA-seq dataset. Samples in this dataset are gene expression vectors and the task is to classify types of cancer. This dataset is part of the cancergenome project [7].

- Text dataset (Lang, 1995) containing newsgroup articles, where features are created using word frequencies and the goal is to classify the type of document.

- Speech dataset from the TIMIT corpus (Garofolo, 1993). The feature vectors are mel-frequency cepstral coefficients, and the task is vowel classification.

Using these datasets, Sharma et al. train two-dimensional CNNs and find that, after fine tuning hyperparameters, the DI pipeline outperforms current benchmarks on the three classification tasks.

The DI pipeline can be split into two main steps. The first step consists of building a binary image from the *entire* training set in an unsupervised manner. This binary image distinguishes pixels that correspond to one or more features (white pixels) from pixels that don't (black pixels). For the reasons mentioned

[6] *Two dimensional* refers to the shape of the input being described by two numbers.
[7] https://cancergenome.nih.gov

in the motivation section, this binary image -or mask- should capture the high-dimensional structure of the dataset. For example, it could cluster similar features while separating different ones. The second step is to assign a grey-scale value to each white pixel in the image for each sample in the dataset. Consequently, the transformed samples are versions of the same binary image but *coloured* according to the values in the original sample. From a genomics point of view, we first build an image-like representation of the whole population taking into account the similarity between different SNPs and then particularize this representation to each individual according to its genome.

Let $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ be the training set, where samples x_i have dimension p. In a genome-enabled prediction context p corresponds to the number of SNPs. The columns of this dataset can be viewed as feature vectors $\{g_1, g_2, ..., g_p\}$. In order to create the matrix representation mentioned in the previous paragraph, feature vectors are mapped into a 2-D space.

These embeddings are computed using one of two unsupervised dimensionality reduction techniques: kPCA (kernel Principal Component Analysis) or t-SNE (t-distributed Stochastic Neighbor Embedding). On one hand, kPCA is an extension of PCA where the linear projection is performed in a reproducing kernel Hilbert space (Schölkopf, Smola, & Müller, 1998). On the other hand, t-SNE learns a low-dimensional mapping $\{\hat{g}_1, \hat{g}_2, ..., \hat{g}_p\}$ that preserves the similarities between the original feature vectors (Van der Maaten & Hinton, 2008). Specifically, similarities in the high-dimensional space are computed as:

$$sim_{ji} = p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

where

$$p_{j|i} = \begin{cases} \frac{\exp\left(-\|\mathbf{g}_i - \mathbf{g}_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|\mathbf{g}_i - \mathbf{g}_k\|^2 / 2\sigma_i^2\right)} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

σ_i being a learnable parameter. Likewise, similarities q_{ij} in the low-dimensional space are measured using the Cauchy distribution:

$$q_{ij} = \frac{\left(1 + \|\hat{\mathbf{g}}_i - \hat{\mathbf{g}}_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} \left(1 + \|\hat{\mathbf{g}}_k - \hat{\mathbf{g}}_l\|^2\right)^{-1}}, \quad \text{for } i \neq j,$$

Then, the embeddings $\hat{g}_i$ are learned, via gradient descent, to minimize the Kullback-Leiber divergence between the distribution of similarities:

$$\mathrm{KL}(P \| Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Each embedding $\hat{g}_i$ is treated as the coordinates of the feature i on a Cartesian plane, and the feature set $\{\hat{g}_1, \hat{g}_2, ..., \hat{g}_p\}$ forms a point cloud in this plane. After locating each feature in the plane, the smallest rectangle containing the point cloud is found. Convex Hull algorithm is used to accomplish this task. Following

a rotation to align the base of the bounding rectangle with the x axis, the Cartesian planes is discretized. Several features can be mapped to the same pixel. In this case, their values are averaged, which leads to greyscale images, as opposed to binary masks. This pipeline is depicted in Figure 6.9.

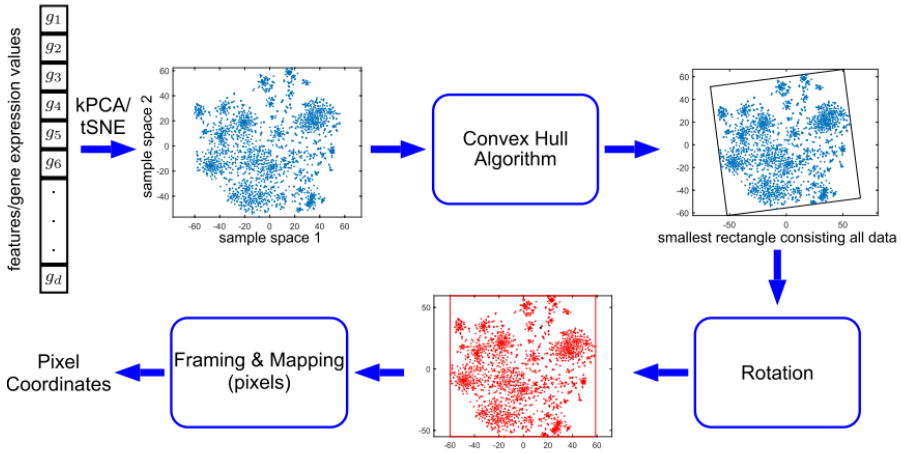
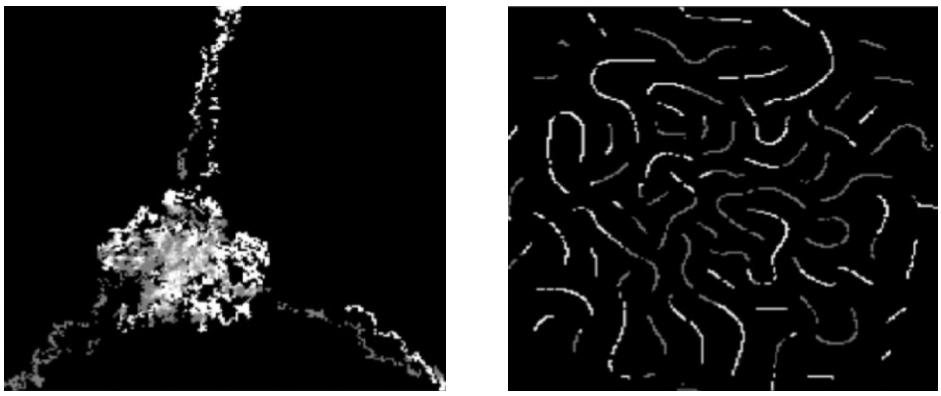


Figure 6.9: Genome to Image pipeline from Sharma et al. (2019)

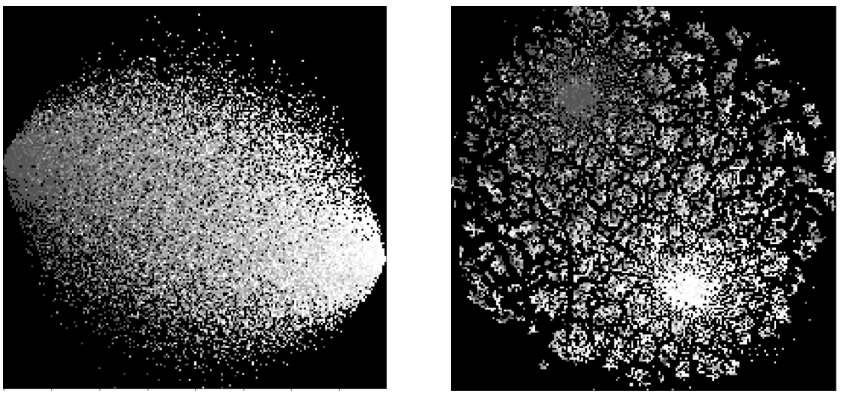
Applying this pipeline yields the results shown in figures 6.10 and 6.11.



(a) kPCA (b) t-SNE

Figure 6.10: 200×200 Image representation of a yeast genome using the DeepInsight pipeline.

The datasets obtained with the DI method were used to train 2D CNNs. Similarly to the 1D CNNs described in Section 6.1, their architecture is vaguely inspired by AlexNet. Hyperparameter tuning was tackled via randomized search. The resulting architectures are described in diagrams 6.12 and 6.13.



(a) kPCA (b) t-SNE

Figure 6.11: 200×200 Image representation of a holstein genome using the DeepInsight pipeline.

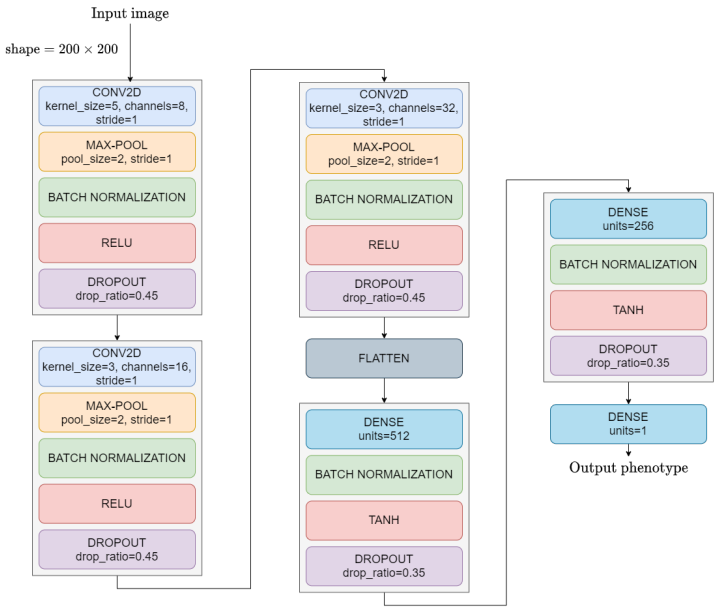


Figure 6.12: 2D CNN architecture in Yeast.

6.4.3 Random mapping

In order to evaluate the extent to which t-SNE and k-PCA capture the high-dimensional structure of the yeast and holstein genome, these mappings are compared to a random mapping. Same as in the original pipeline, if various features are mapped to the same pixel, the values of those SNPs are averaged. The comparison is done in terms of the predictive accuracy in the downstream regression task. That is, a CNN with constant architecture is trained and fine-tuned with the three image datasets: random, t-SNE and kPCA.

The random mapping of SNPs to pixels is done by sampling the X and Y

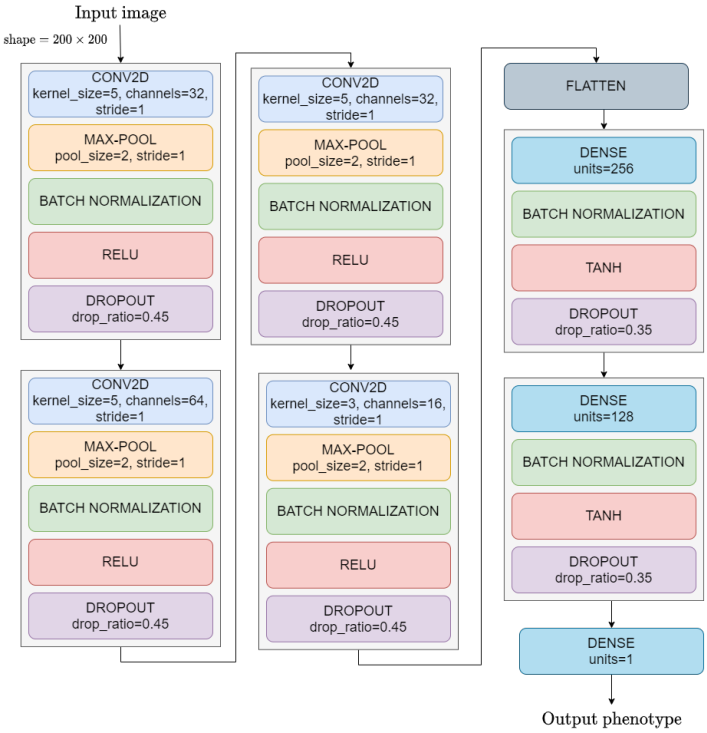


Figure 6.13: 2D CNN architecture in Holstin

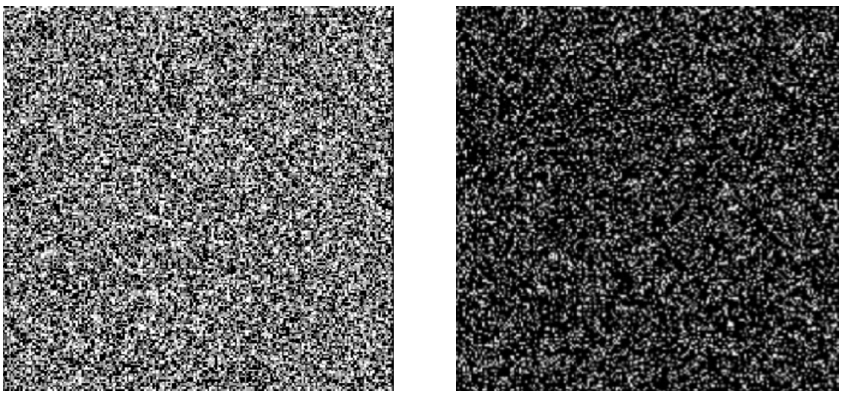
coordinates of each feature from a uniform distribution (i.e: $X \sim \mathcal{U}[0, 199]$, $Y \sim \mathcal{U}[0, 199]$). As was done in the other mapping schemes, if two or more features are mapped to the same pixel, their values are averaged.

Using this mapping scheme, the position of each feature in the plane is random, and thus, irrelevant to the regression task. No spatial information is present in the random image. We expect CNNs trained on t-SNE and kPCA to outperform the CNN trained on random image representations of the genome.

6.4.4 Fermat Distance

Motivation

In dimensionality reduction, a notion of similarity between points in high-dimension is needed. For instance, in t-SNE, euclidean distance between feature vectors was used to construct two-dimensional embeddings of $\{g_1, g_2, ..., g_p\}$. However, the choice of a similarity measure is not simple. A well-studied phenomenon in machine learning is the so-called *curse of dimensionality*. Beyer, Goldstein, Ramakrishnan, and Shaft (1999), illustrate this phenomenon by analyzing the effect of dimensionality increase on the distances between data points. They argue that, under a broad set of conditions, as dimensionality increases, distances between every two points tend to be equal. Moreover, they maintain the view that for large distances the distortion is more notorious than for small ones. Aggarwal,



(a) Holstein (b) Yeast

Figure 6.14: 200×200 Image representation of a Holstein and a Yeast genome using random mapping

Hinneburg, and Keim (2001), emphasize this argument, suggesting that in high dimensional spaces, L_k norms may not be qualitatively meaningful. Therefore, dimensionality reduction algorithms, such as t-SNE, may benefit from using a different, non-euclidean notion of similarity, more adequate for high dimensional datasets.

Sapienza, Groisman, and Jonckheere (2018) propose a notion of similarity called *Weighted Geodesic distance following Fermat's principle*, or simply *Fermat distance*, that aims to capture both the structure of the manifold on which the data lies.

Definition

Let $\mathbb{X}_n$ be a sample of n independent vectors with common probability density f. In our case the elements of $\mathbb{X}_n$ are genomic sequences of dimension p. Let $\mathcal{M} \subseteq \mathbb{R}^p$ be a d dimensional manifold, with $\mathbb{X}_n \subseteq \mathcal{M}$ and $d \leq p$.

The Fermat Distance between two points x_1, x_2 is defined as:

$$D(x_1, x_2) = \inf_{\Gamma} \int_{\Gamma} \frac{1}{f^{\beta}} d\ell$$

where $\beta > 0$ and the infimum is taken over all the paths Γ that connect x_1 with x_2. As shown in the equation above, Fermat distance corresponds to a density-weighted shortest path between two points. Paths where the density is high are favoured in the minimization. As explained in (Sapienza et al., 2018) this is a *weighted geodesic distance* where the path is weighted inversely to the density f.

In practice, f is unkwnown, and only the set of vectors $\mathbb{X}_n$ sampled from f is observed. Thus, Sapienza et al. (2018) propose an estimator of the Fermat distance.

$$D_{\mathbb{X}_n}(x_1, x_2) = \min_{\substack{(q_1, \ldots, q_K) \in \mathbb{X}_n^K \\ q_1 = x_1, q_K = x_2}} \sum_{i=1}^{K-1} \ell\left(q_{i+1}, q_i\right)^{\alpha}$$

where $\alpha = d\,\beta + 1$ [8], $2 \geq K \geq n$ and $\ell(\cdot, \cdot)$ is a distance on $\mathcal{M}$ (for instance, the Euclidean distance). Thus, in the Fermat distance estimator (or sample Fermat distance), the minimization is done over the finite sequences of points in $\mathbb{X}_n$ that go from x_1 to x_2. Note that when $\alpha = 1$, the distance $\ell(\cdot, \cdot)$ is recovered, since the sequence that yields the shortest sample Fermat Distance will be (x_1, x_2). However, when $\alpha > 1$, consecutive points with large $\ell(\cdot, \cdot)$ are discouraged, and the sequence tends to follow regions with high density and only include points that are close. In that sense, the Fermat sample distance follows more closely the structrue of the manifold on which the data lies than the original $\ell(\cdot, \cdot)$ distance. Sapienza et al. (2018) and Groisman, Jonckheere, and Sapienza (2018) show that the sample Fermat Distance is indeed a distance (positive, symmetric and verifies the triangular inequality) and provide conditions for the convergence of the sample to the original Fermat Distance.

Fermat and t-SNE in the DI pipeline.

As explained in Section 6.4.2, the version of t-SNE used to create low-dimensional embeddings employs the euclidean distance between points in high dimension. As suggested by Sapienza et al. (2018), replacing the Euclidean distance by Fermat distance, could be beneficial in clustering or dimensionality reduction tasks. We trained 2D-CNNs on images built by applying de DeepInsight pipeline with t-SNE, replacing Euclidean distance by Fermat distance. The results of this experiment are presented in section 6.5.2.

Sapienza et al. (2018) propose using the Floyd-Walsh algorithm (Floyd, 1962) to compute the sample Fermat distance in $O(N^3)$ operations. Nonetheless, in applications where the number of points is high, this implementation was still computationally expensive. Therefore, the authors also implement an algorithm to compute an approximate sample Fermat distance [9]. The complexity of this algorithm is $O(lkn \log n)$, where l and k are typically much smaller than n. This enabled the use of Fermat distance to construct the image-like embeddings of the samples in the Yeast dataset.

6.5 Results and Discussion

In this Section we present the results of both the 1D and 2D CNNs for the two chosen datasets (Yeast and Holstein), comparing the performances between differ-

[8] The choice of β depends on the application but Sapienza et al. (2018) give certain guidelines on how to choose it.

[9] This implementation can be found in: https://github.com/facusapienza21/Fermat-distance

ent experiments. We also delve into the 1D CNN's saliency maps and the most relevant 1D CNN's activation maps, proposing possible causes for their differences in performance. We evaluate the suitability of CNN's inductive biases and the DI algorithm for this particular prediction task. Lastly, we conclude with a global comparison between these Deep Learning methods and the classical alternatives presented in Chapter 4.

6.5.1 1D CNNs

Yeast

For the Yeast dataset, we consider the Lactate, Lactose, Xylose and Sorbitol environments. As mentioned in Section 3.1.1, these are sugar-related environments which encourage Yeast growth, and as such they enjoy mid to high pairwise correlations.

Overall results As mentioned in Section 6.3.5, we distinguish four different CNN architectures: the base AlexNet-like CNN and the residual CNN, with their corresponding single and multi-trait variants. In addition, we compare the highest-performing CNN to a 5-layer MLP described in Table 6.2.

Figures 6.15 shows the test performance (in terms of Pearson's correlation) on four different Yeast environments. The results indicate that residual CNNs outperform the AlexNet-like architectures on all four environments, with both single and multi-trait variants achieving best and second-best performance. The difference between multi and single-trait variants is small, specially for the residual CNNs.

Regarding the comparison to the MLP architectures, the CNNs outperform them in all environments except Lactate, and they do so by a considerable margin. In Lactate, the gap between the best CNN (Residual single-trait) and the MLP is practically nil. This could indicate that the CNNs inductive biases are relevant for the traits' prediction on most environments.

A global comparison between the performances of all models in the Yeast dataset is shown in Table 6.3.

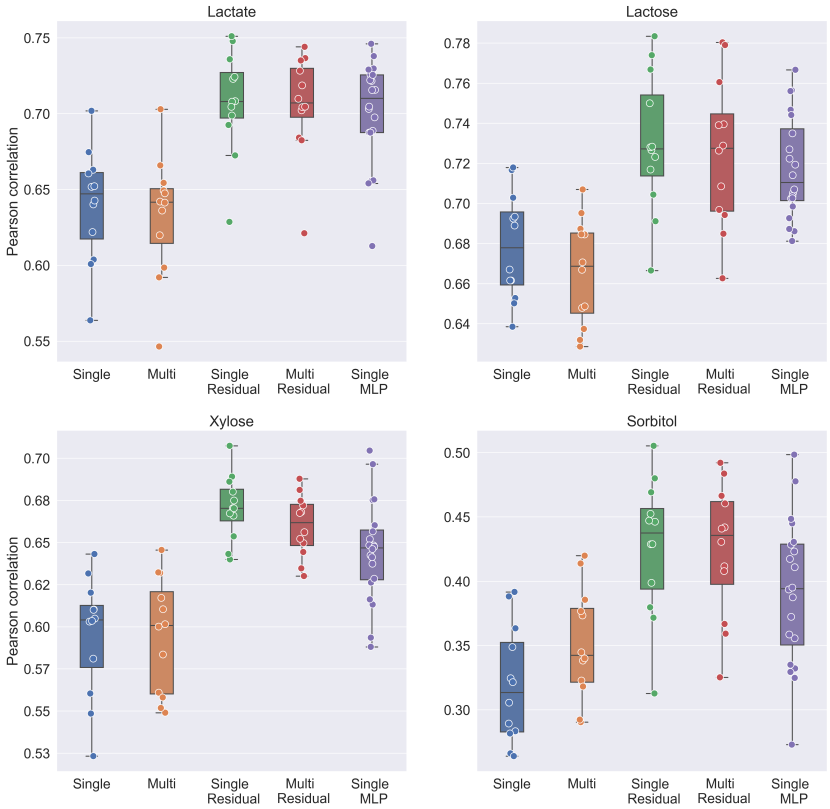


Figure 6.15: 1D CNN and MLP results on four environments in the Yeast dataset.

Marker shuffling To verify if the CNNs inductive biases are responsible for their superior performance w.r.t the MLPs, we conducted a series of marker shuffling experiments. The results of these experiments are shown in Figure 6.16.

When shuffling markers, any kind of spatial structure in the input signal is lost. Revisiting the analogy with computer vision, shuffling an image's pixels would make it unrecognisable, making any CNN's performance suffer greatly. This does not occur in the yeast genome-wide phenotype prediction task; instead, the performance of the shuffled CNNs increases. This may suggest that the CNN is not exploiting the local structure in the signal or that the model's translational equivariance is not appropriate for the task at hand.

We naturally wonder why this phenomenon occurs. To gain more insight into this question, we explored the network's saliency maps.

It is worth noting that all subsequent plots involving the shuffled CNNs are showed after undoing the marker shuffling transformation. We will also make extensive use of channel-wise sums to help visualize activation maps. This does not mean that the residual CNNs perform a channel-wise sum - they flatten the addition layer's output instead. Still, if optimal, channel-wise sums can be easily performed by the network after the flatten operation.

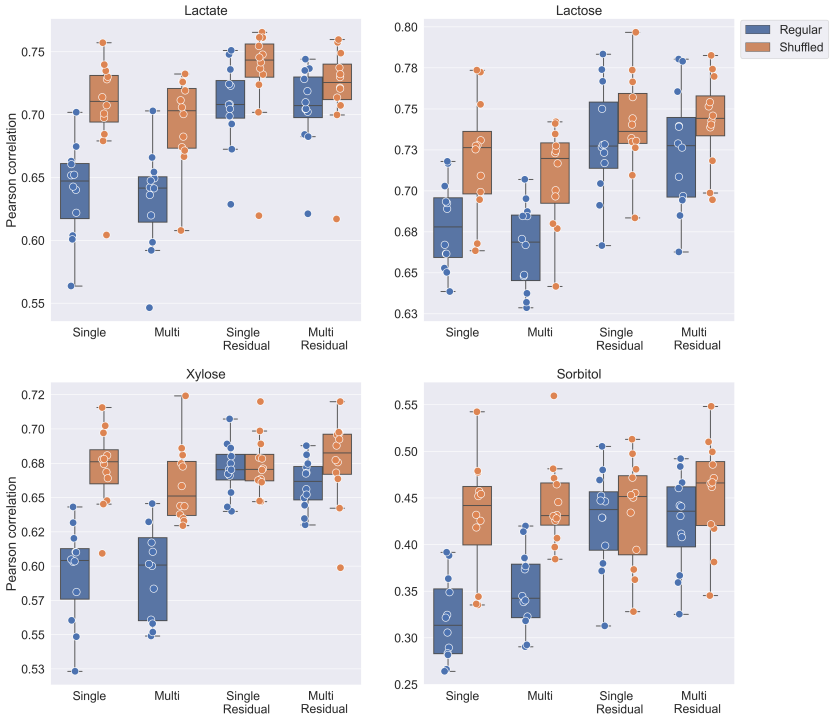


Figure 6.16: Comparison between regular and shuffled 1D CNNs in the Yeast dataset.

		Lactate	Lactose	Xylose	Sorbitol
CNN	Multi	0.63	0.67	0.60	0.35
	Multi Residual	0.71	0.73	0.66	0.42
	Multi Shuffle	0.69	0.71	0.66	0.44
	Multi Shuffle Residual	0.72	**0.74**	**0.68**	**0.46**
	Single	0.64	0.68	0.59	0.32
	Single Residual	0.71	0.73	0.67	0.43
	Single Shuffle	0.71	0.72	0.67	0.43
	Single Shuffle Residual	**0.73**	**0.74**	0.67	0.43
MLP	Multi	0.71	0.72	0.66	0.42
	Single	0.70	0.72	0.64	0.39

Table 6.3: Mean CNNs and MLPs performance on the yeast dataset. The bold numbers indicate the best performing model for each environment.

Saliency maps As explained in Section 6.3.7, saliency maps indicate the magnitude of the gradient of the output with respect to each feature of an input sample:

$$s = \nabla_x(f(x))$$

where f is the model and x an input sample.

Figure 6.17 shows the saliency maps for different models trained on the yeast dataset (marker weights for Ridge). We can observe that the impact of each marker

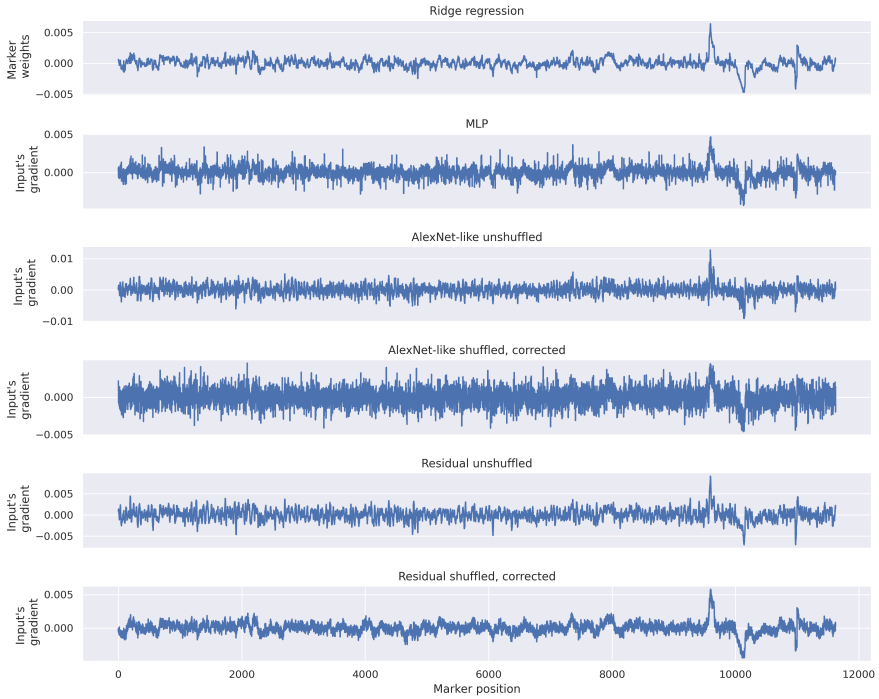


Figure 6.17: Different models' mean (averaged over all test samples) saliency maps in the yeast dataset. The first plot corresponds to the weights of a Ridge regression. The corrected shuffled plots are the gradients of the shuffled CNN after undoing the random marker shuffling transform.

to the final prediction is extremely similar across models. Specifically, the region around SNP 10,000 appears to be the most relevant region in all models. The deep learning models exhibit maps which closely resemble a regularized linear regression. Moreover, the unshuffled and shuffled CNNs end up learning the same marker importances. The main difference between these two models is that the unshuffled CNN's saliency map is noisier. When comparing the AlexNet-like architecture to the residual one, the former exhibits a noisier saliency map in both the regular and shuffled variants. In addition, unlike the residual architecture in which shuffling the markers smoothes the input's gradients, we observe the opposite effect. The similarity between the saliency maps and the weights of the Ridge regression suggests that the CNN's inductive biases do not affect feature importance significantly.

Activation maps Figures 6.18 and 6.19 show the channel-wise output of the residual and non-residual branches of the CNN (just before the addition layer), for the regular and shuffled variants. Similarly, Figures 6.20 and 6.21 show the residual and non-residual branches' channel-wise sum. The biggest difference between the regular and shuffled CNNs relies on the non-residual branch. This is further confirmed by Figure 6.22, which shows that both networks' residual representations

come extremely close to perfect identity mappings.

In the unshuffled case, the non-residual branch acts as a "edge detector" , in the sense that its largest activations correspond to abrupt changes in the input signal. Unsurprisingly, its shuffled counterpart is unable to distinguish these spatial patterns, something that is reflected on its noisy activation maps.

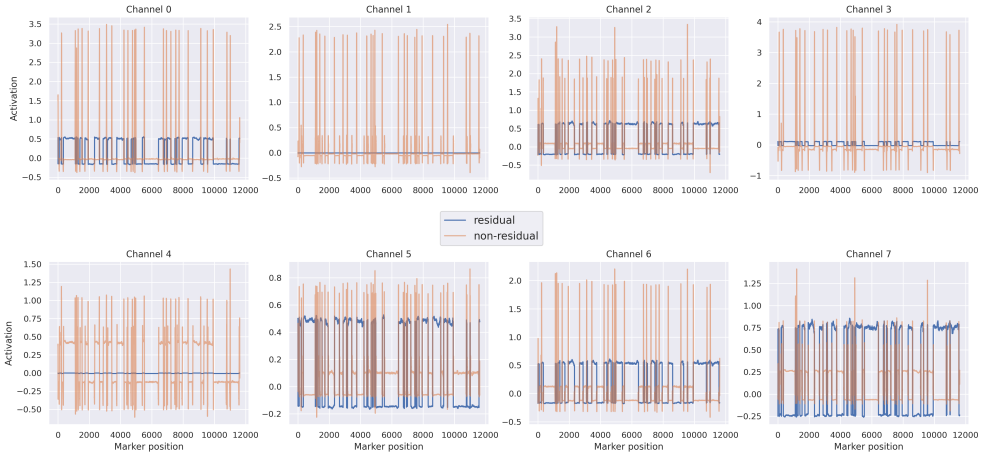


Figure 6.18: Superimposed residual and non-residual branches' output for the unshuffled variant (one sample only, not to be confused with the previous mean across all individuals). Recall that each channel corresponds to the output feature map of a single convolutional filter.

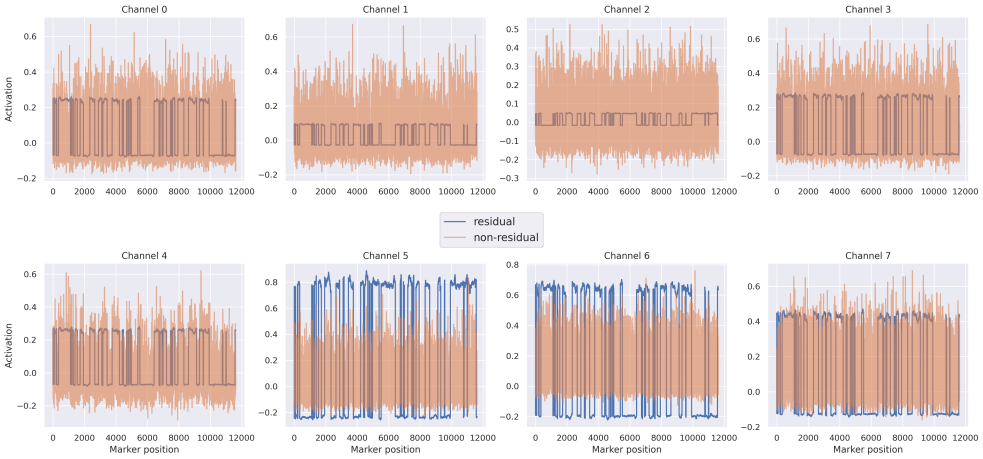


Figure 6.19: Superimposed residual and non-residual branches' output for the shuffled variant.

Lastly, we analyse the addition layer's output, which combines the residual and non-residual branches and is fed to the MLP. Figures 6.24 and 6.23 showcase the addition layer's channel-wise sum. It can be observed that this channel-wise sum is a slightly distorted version of the original signal. In other words, the convolutional

Figure 6.20: Residual and non-residual branches' channel-wise output sum for the unshuffled variant.

Figure 6.21: Residual and non-residual branches' channel-wise output sum for the shuffled variant.

layers learned a mapping that resembles an identity. This suggests that the model may have learned to ignore the convolutional layers, to base its predictions on the final fully connected layers.

The shuffled CNN can recover a closer representation to the original input signal than its unshuffled counterpart (due to the peaks in the amplitude jumps). Moreover, this representation is enhanced with finer, higher frequency information, which come from the non-residual branch (as seen in Figure 6.21). This is not the case for the unshuffled CNN, in which the constant (non-peak) regions of the learnt

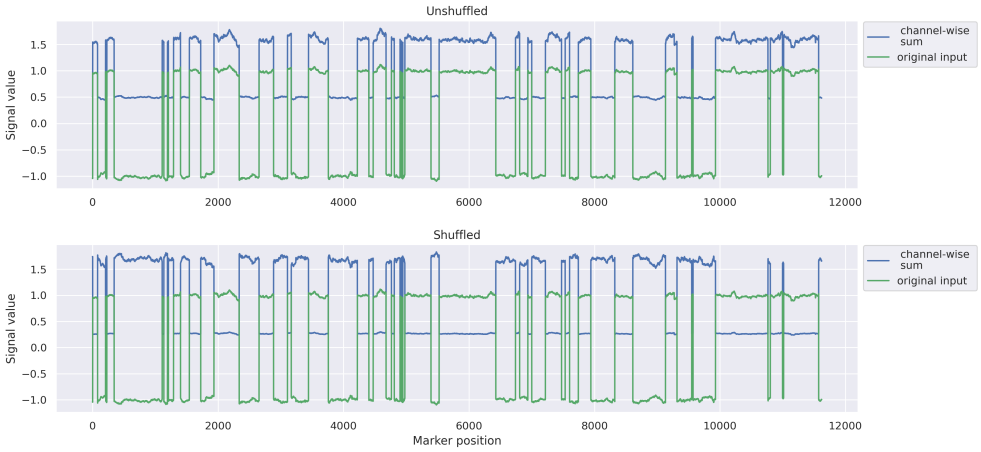


Figure 6.22: Comparison between the CNN's residual representations. Both channel-wise sums come extremely close to the original input signal, differing in a slight offset and an inversion in the case of the shuffled variant.

representation are virtually identical copies of the input signal.

Shuffling the marker order may allow the CNN to aggregate markers from different regions of the genome. Furthermore, the unshuffled CNN aggregates large chunks of constant input signal (due to this particular dataset's characteristics). We hypothesize that this leads to a poor representation of the input genotype, which is largely governed by the peaks that arise from amplitude jumps. This kind of edge-detecting features could be useful in certain tasks, but the empirical results indicate they are not useful in the context of yeast genome-wide complex genotype prediction. Instead, the results favour a representation which is more sensitive towards each individual marker's contribution, i.e. the shuffled CNN's. In other words, the position in which a certain pattern occurs affects its outcome, which directly contradicts the translational equivariance of CNNs. This observation is consistent with our knowledge of the inner-workings of the DNA.

This phenomenon may also explains why residual CNNs consistently outperform AlexNet-like architectures. The former can learn representation which resembles the input genotype more easily than the latter. In fact, learning perturbations around identity mappings were one of the main motivations for the introduction of residual layers in (He et al., 2015a).

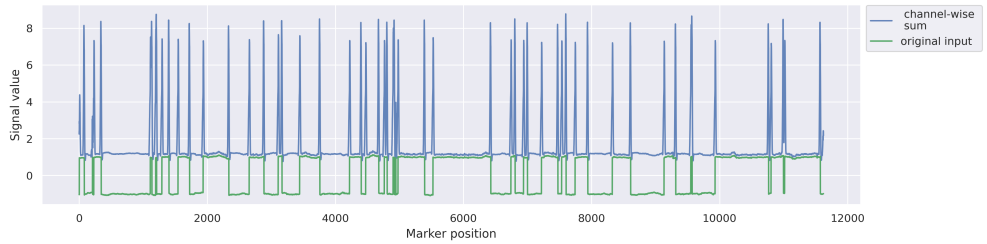


Figure 6.23: Unshuffled CNN final activation map.

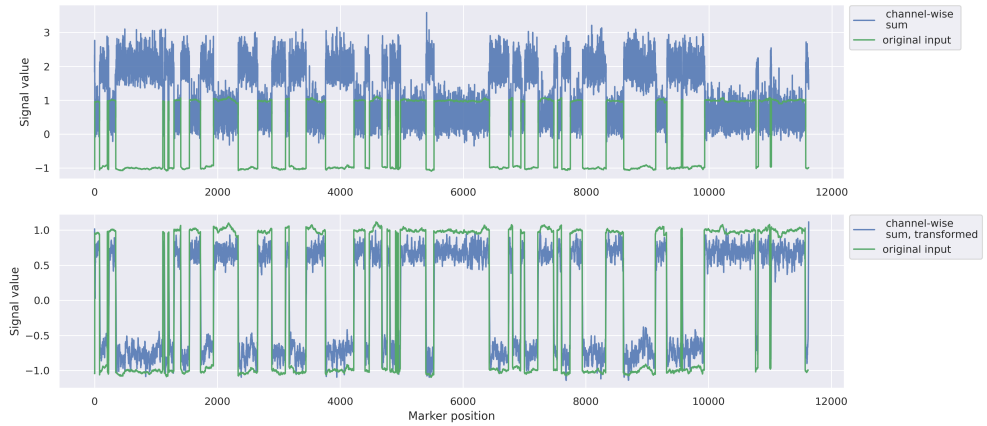


Figure 6.24: Shuffled CNN final activation map. The transformation on the bottom plot consists of an inversion, an offset correction ($+1.25$) and a moving average (10 sample window length).

Holstein

We consider all three traits in the Holstein dataset, namely Somatic Cell Score (SCS), Milk Fat Percentage (MFP), and Milk Yield (MY). The only correlated traits in this datasets are MY and MFP, as shown in Section 3.1.4.

Overall results Figures 6.25 shows the different CNN's and the MLP's results for the three different Holstein environments. The mean results can be found in Table 6.4.

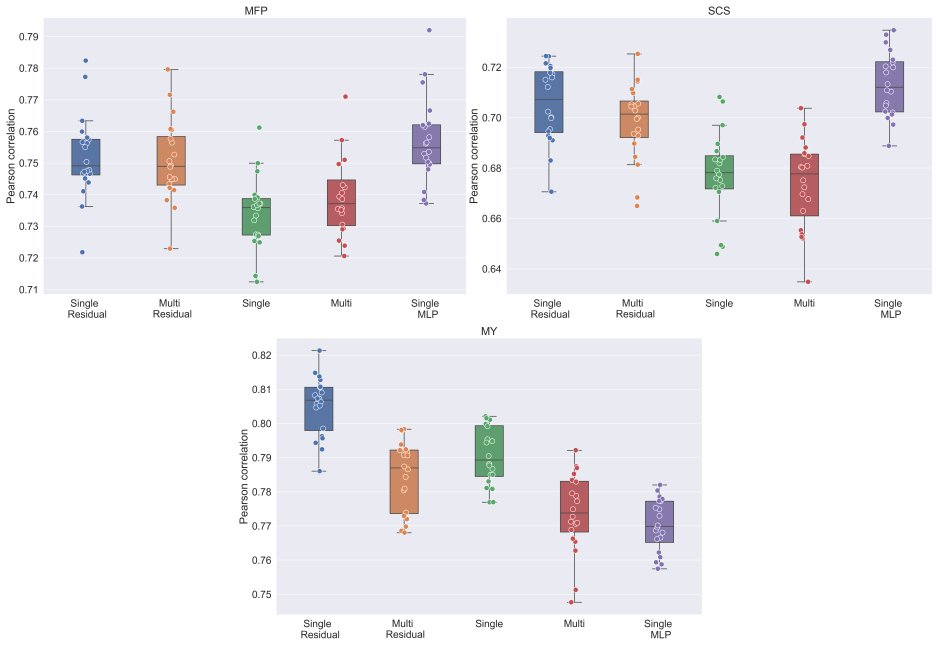


Figure 6.25: 1D CNN and MLP results in all three traits from the Holstein dataset.

The results were extremely similar to the yeast dataset for the SCS and MFP traits. Namely, the residual CNNs consistently outperformed the AlexNet-like architectures, with little difference between multi and single-trait approaches. However, in this dataset, the best CNN architectures (single-trait residual) for these two traits performed slightly worse than the MLP.

The MY trait exhibits a different behaviour. Although the best performing CNN is still the residual one, the AlexNet-like architecture achieved second place, with both networks belonging to the single-trait variants. In addition, the performance margin between the best CNN and the rest of the models is considerably larger than in other traits.

It is worth noting that MFP and MY have the highest pairwise correlation. However, SCS and MFP results presented in this section were similar, whereas MY showed a different behaviour. This was most prominent for the MLP vs CNN comparison, and also for multi-trait models, where MY showed a large performance drop.

Marker shuffling The large performance gap between the CNN and MLP in MY suggests that the CNN's inductive biases may be beneficial in this trait. The marker shuffling ablation study was conducted to test this hypothesis.

Figure 6.26 shows the results of the marker shuffling experiments for each Holstein trait. In addition, the comparison between all models' mean performance is showed in Table 6.4. Similarly to the Yeast experiments, marker shuffling does not impact MFP's and SCS's predictions significantly. Moreover, some models benefit from marker shuffling, a phenomenon we already observed in the yeast dataset.

The most interesting result comes from the MY trait, in which marker shuffling consistently hinders performance on all four models, and it does so by a significant margin. The consistent, large gap between unshuffled and shuffled CNNs is a strong indicator that the CNN's inductive biases are adequate for this trait. Namely, in this trait, detecting patterns in the genome regardless of their absolute position may be relevant.

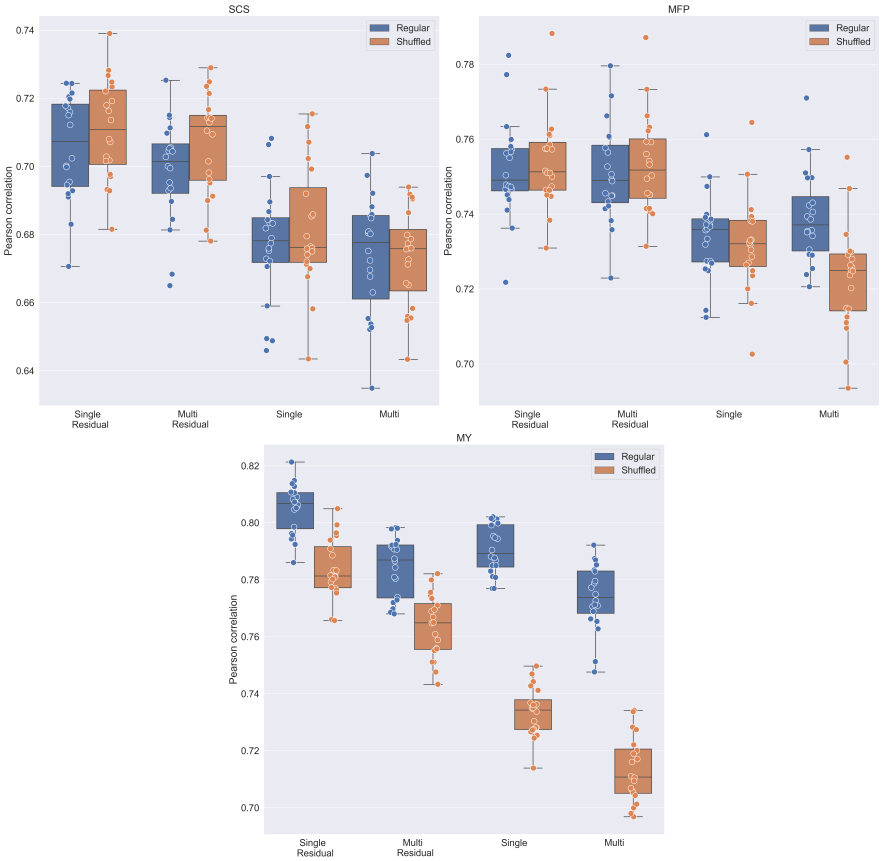


Figure 6.26: 1D CNN marker shuffling results in the Holstein dataset.

		SCS	MY	MFP
CNN	Multi	0.74	0.77	0.67
	Multi Residual	0.75	0.78	0.70
	Multi Shuffle	0.72	0.71	0.67
	Multi Shuffle Residual	0.75	0.76	0.71
	Single	0.73	0.79	0.68
	Single Residual	0.75	**0.81**	0.71
	Single Shuffle	0.73	0.73	0.68
	Single Shuffle Residual	0.75	0.78	0.71
MLP	Multi	**0.76**	0.77	**0.72**
	Single	**0.76**	0.77	0.71

Table 6.4: Mean CNNs and MLPs performance on the Holstein dataset.

Saliency maps The MY trait's results motivate us to conduct the same saliency and activation map analysis as in the Yeast section. This analysis will be done on single-trait CNNs, which achieved superior performance in this particular trait.

Figure 6.27 shows a Ridge regression's learned marker weights alongside the MLP's and CNN's input's gradients. As in the Yeast dataset, most gradients exhibit a consistent, well defined pattern, which closely resembles the linear regression's weights. For this trait's prediction, the most relevant region is located around marker 25,000, with a smaller peak around marker 11,000.

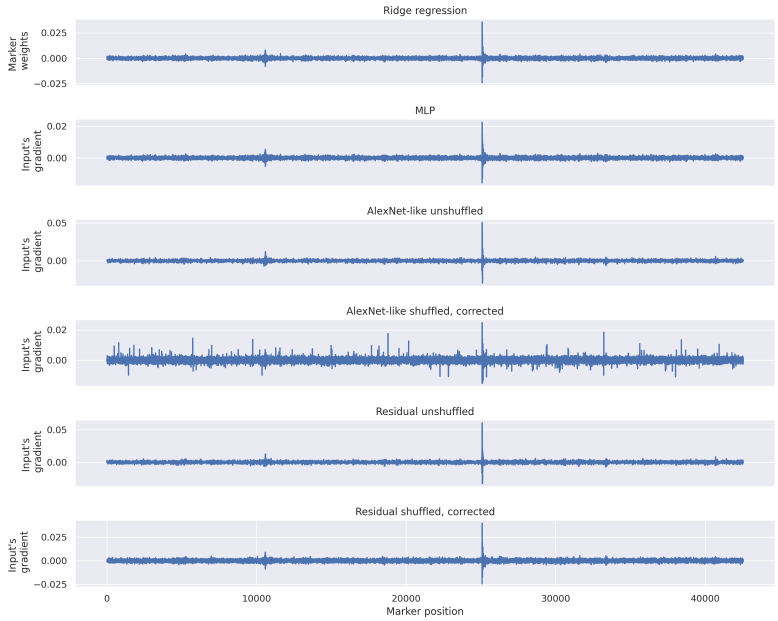


Figure 6.27: Ridge's regression marker weights and mean saliency maps for the MY trait.

The only model which escapes this pattern is the shuffled AlexNet-like CNN, which exhibits several gradient peaks along the entire input. In contrast, the shuffled residual CNN's does not manifest these irregular saliency maps. Since this network performed worse than the Residual single-trait CNN, we can assume that this irregular gradients do not correspond with better performance. Moreover, the shuffled Residual CNN's performance is considerably higher than its AlexNet-like counterpart.

The shuffled Residual CNN's gradients are slightly noisier than its unshuffled counterparts', something that also occurred in yeast's shuffled residual CNN.

Activation maps It is worth noting that Holstein's activation maps are considerably harder to interpret than Yeast's. This is due to the former's irregular input signal when compared to latter's square-wave-like shape. Still, by examining their relative magnitude and perturbations w.r.t. the input, the activation maps shed some light on the relevance of the CNN's Residual connection.

Figures 6.28 and 6.29 show the residual and non-residual branches' activation maps channel-wise sum. The magnitude of activations is comparable for both branches, which implies the model is taking advantage of the Residual block. The fact that Residual CNNs outperformed both the MLP and the shuffled CNNs in this trait suggests that the bias introduced by the identity branch is useful for the prediction of this particular trait.

Figure 6.28: MY's residual and non-residual branches' channel-wise output sum for the unshuffled variant.

Figure 6.30 shows the CNN's channel-wise sum after the addition layer, superimposed with the original input signal. In contrast to Yeasts' final activations, Holstein's show slight qualitative similarities with the original input. This is consistent with the fact that the CNN trained on shuffled samples did not outperform

Figure 6.29: MY's residual and non-residual branches' channel-wise output sum for the shuffled variant.

the unshuffled version, something which did happen in the Yeast dataset. Moreover, the input's negative markers are set to almost zero. This is likely due to the LeakyReLU non-linearity, however the channel-wise sum does not necessarily need to meet this criteria.

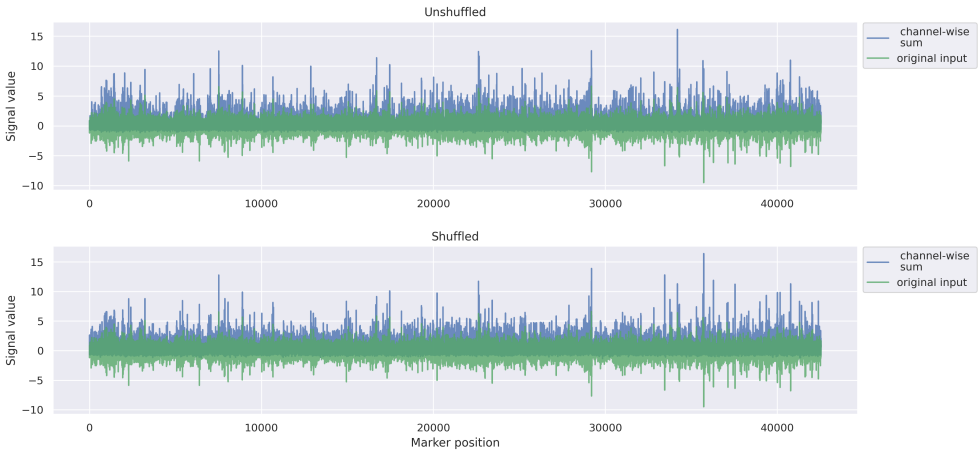


Figure 6.30: MY's comparison between the CNN's final representations.

6.5.2 2D CNNs

Results shown in the following section represent the mean predictive correlation (r) in 6 data splits (train-validation-test).

Yeast

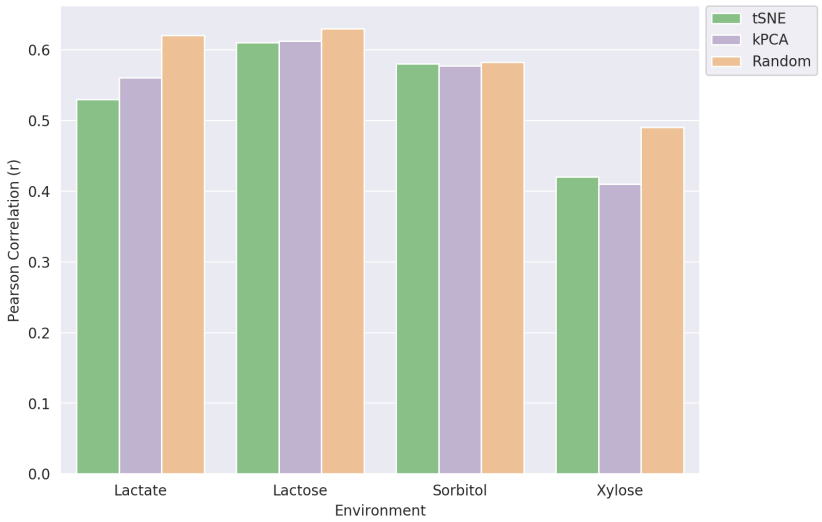


Figure 6.31: Performance comparison of 2D CNNs after the DI pipeline with two different mappings in Yeast.

As shown in Figure 6.31, in all environments, the difference between k-PCA and t-SNE mappings is not significant. As mentioned in Section 6.4.3, the goal of the random mapping experiments is to evaluate the extent to which t-SNE and k-PCA capture the high-dimensional structure of the data. Figure 6.32 shows that random mappings are not detrimental in terms of predictive accuracy. On the contrary, in Yeast, the 2D CNN trained on images built by randomly mapping SNPs to the plane outperformed models that use t-SNE and k-PCA. This suggests that, in the case of the yeast genome, the increase in performance with respect to the 1-D CNNs [10] is not due to the DeepInsight pipeline.

As shown in Figure 6.32, using Fermat distance to build two-dimensional feature embeddings was beneficial, surpassing random mapping and the literature in three out of four environments. This indicates that, in terms of predictive accuracy in the downstream phenotype regression task, Fermat Distance may be a more effective measure of similarity than Euclidean distance for high dimensional genotypes. Whether this result can be extrapolated to other dimensionality reduction techniques, models and traits is subject of further work.

Nevertheless, the CNNs trained on the images generated using the DI pipeline did not outperform some baseline models 6.5.3.

[10]The comparison is shown in Table 6.5.3.

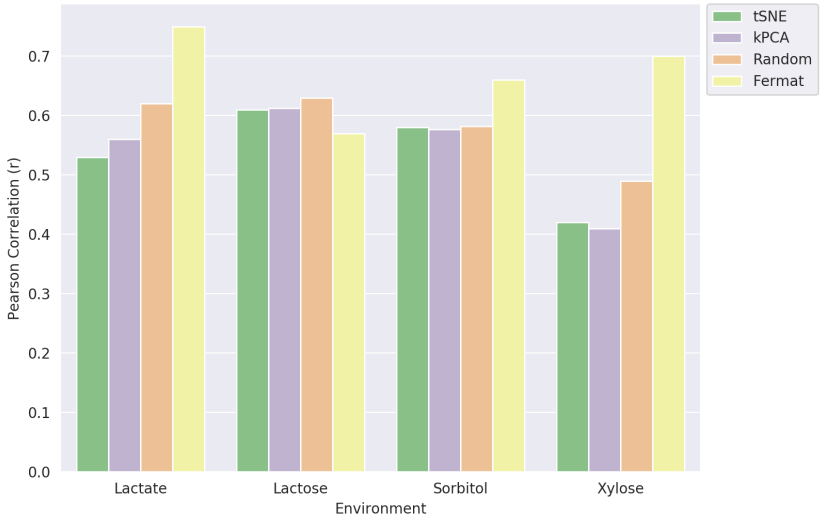


Figure 6.32: Performance comparison of 2D CNNs after the DI pipeline with four different mappings, including Random and Fermat in Yeast.

Holstein

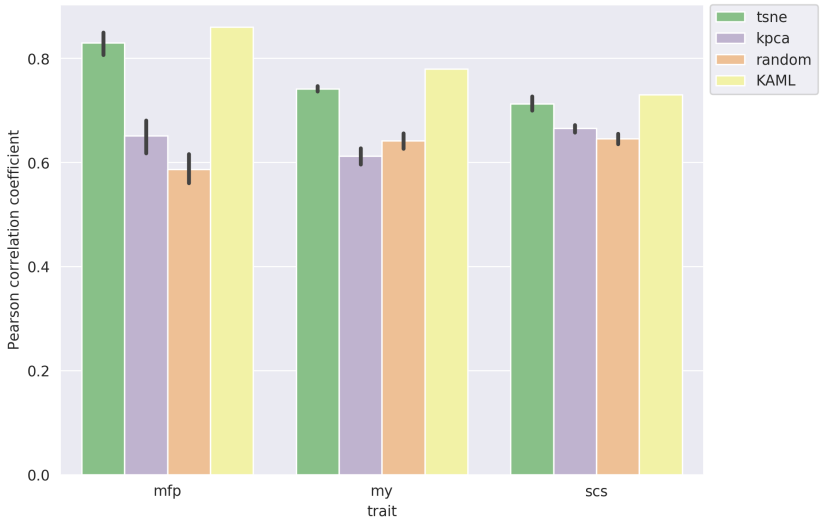


Figure 6.33: Performance comparison of 2D CNNs after the DI pipeline with three different mappings in Holstein.

In Holstein, CNNs trained on images built with t-SNE consistently outperfom their counterparts built by applying the DI pipeline with random and k-PCA mappings. The 2D-CNNs do not outperfrom the bayesian mixed linear model presented in (Yin et al., 2020). However, KAML outperforms the 2D-CNNs by

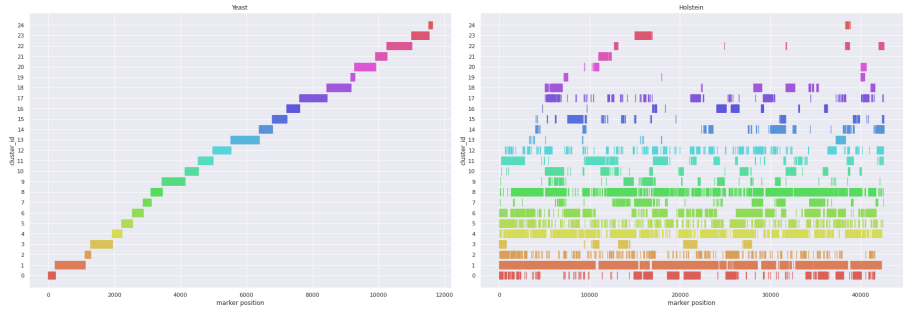
less than 4% in all traits. Furthermore, the 2D-CNN outperforms all baseline models in Milk Fat Percentage. This may suggest that the local structure of the image-like representation built using DI and t-SNE is exploited by the CNN via convolutions.

In contrast to the Yeast experiments, t-SNE consistently outperforms k-PCA in Holstein. It can be pointed out that, albeit locally, t-SNE might better preserve euclidean distances (by preserving probabilities). However, whether euclidean distances in the high dimensional input space are informative - and useful for the task at hand - is unclear, due to the *curse of dimensionality* and the fact that the variables are discrete. In addition, when using t-SNE non-zero pixels were more spread within the image (see Figure 6.11), which may be caused by the asymmetry of the KL divergence loss (Van der Maaten & Hinton, 2008). Further research is needed to evaluate whether these aspects favour CNNs trained on those feature maps.

As shown in Figure 6.33, the variance in predictive correlation is higher (almost double) in MFP than in SCS and MY. This may be linked to the genetic architecture of the traits. Milk Fat Percentage has one or several major genes with large effect and many loci with small efect, while SCS and MY only have several loci with small or moderate effect (Yin et al., 2020).

On DeepInsight

The results outlined in the previous paragraph indicate that the DeepInsight method worked better in the Holstein dataset than in the Yeast dataset. This may be due to the genetic architectures of these traits.



(a) Yeast (b) Holstein

Figure 6.34: Hierarchical clustering results.

Figure 6.34 shows the result of the hierarchical clustering experiment in the Yeast and Holstein datasets. As explained in Section B.1, clusters represent groups of similar SNPs. Since similarity is defined in terms of correlation, SNPs from a cluster tend to be in linkage disequilibrium. Figure 6.34 shows that, in Yeast, clusters are formed by contiguous SNPS, while in Holstein, SNPs from a same cluster can be further away in the genome. One of the main reasons to apply the Deep Insight method was to create genome representations where similar SNPs

are close. This would enable the aggregation of local information via convolutions. The result of the hierarchical clustering experiment indicates that, in Yeast, an image-like representation has no advantage over the genomic sequence due to the absence of non-local correlations. This suggests that there is more to gain from applying the Deep Insight pipeline and a 2D-CNN in the Holstein dataset than in the Yeast dataset. This observation is consistent with the results of random mapping experiments.

The Deep Insight method has recently gained attention due to the fact that it was used by the winner of the *Mechanisms of Action* (MoA) Kaggle competition [11]. Neither in this competition, or in the original paper (Sharma et al., 2019), random mappings were used as a baseline to assess the feature to pixels mappings' downstream performance. As with genomic prediction experiments, the performance of random mappings against the proposed techniques varied depending on the dataset.

For ringnorm-delve, one of the synthetic datasets used in the original paper, we ran the architecture and hyperparameter search described by the authors and obtained a similar accuracy to TSNE mappings, as shown in table 6.5. Table 6.6 shows that the search resulted in smaller receptive fields and larger $L2$ regularization penalties for random mappings.

	Accuracy
Random	0.98
TSNE	0.99
Decision Tree	0.90
Ada-Boost	0.93
Random Forest	0.94

Table 6.5: Ringnorm delve Accuracy for Deep Insight using TSNE and random mappings. Baseline methods from (Sharma et al., 2019) are included for comparison.

	Receptive Field	Num Filters	L2 regularization
Random	2×9	4	3.1×10^{-7}
TSNE	6×4	4	1.4×10^{-7}

Table 6.6: CNN hyperparameters obtained for different random and TSNE mappings.

In the case of the MoA competition, training the same *Efficient-Net B3* (Tan & Le, 2019) architecture (with the same hyperparameters) using random mappings resulted in an increase in average log loss (the competition's chosen performance metric) from 0.0178 to 0.0191, which was significant for this problem. That is, TSNE did outperform random mappings for this dataset, although this may also be due to hyperparameter settings and the architecture used.

[11] https://www.kaggle.com/c/lish-moa/

All in all, we find that in the light of these results the assessment of the DeepInsight technique requires further work, and in that sense comparing against random mappings is a useful first step.

Summary

- 2D CNNs surpassed baseline models for Holstein in Milk Fat percentage, but fell behind on other phenotypes and most Yeast environments.
- In the Yeast dataset, random mappings performed comparably to k-PCA and t-SNE, but Fermat distance outperforms them.
- In Holstein, t-SNE consistently outperformed k-PCA and random mappings.

6.5.3 Global comparison in Yeast and Holstein

As shown in Table 6.8, in the Yeast dataset, GBM outperforms both 1D and 2D CNNs in three out of four environments. In the Sorbitol environment, the 2D CNN trained using the DeepInsight pipeline slightly outperforms the baseline. As described in section 6.5.1, the fact that baseline models outperform CNNs may be due to the inadequacy of the CNN's inductive biases for this particular problem and trait.

In Holstein, 1D CNNs outperform all other models in two out of three traits. However, in MFP, KAML surpasses CNNs and baselines by a significant margin. This may be due to the fact that MFP exhibits the simplest biological architecture (see Yin et al. (2020)). Bayesian models with strong priors may be more effective in traits with simple biological architectures. In contrast, over-parametrised Neural Netowrks seem to be better suited for traits with more complex biological architectures.

As already stated, the results in Yeast and Holstein point that no model is consistently better in all datasets and traits.

	Best Baseline (GBM)	Best 1D CNN	Best 2D CNN
Lactate	**0.78**	0.73	0.75
Lactose	**0.77**	0.74	0.57
Xylose	**0.75**	0.68	0.70
Sorbitol	0.65	0.46	**0.66**

Table 6.7: Mean Pearson Correlation Coefficient for the best Baseline, 1D CNN and 2D CNN model in the Yeast dataset. Results from Grinberg et al. (2019) are not included because Pearson Correlation is not reported.

	Best Baseline (SVR)	Best 1D CNN	Best 2D CNN	KAML
MFP	0.78	0.71	0.81	**0.86**
MY	0.79	**0.81**	0.78	0.79
SCS	0.74	**0.75**	0.70	0.74

Table 6.8: Mean Pearson Correlation Coefficient for the best Baseline, 1D CNN and 2D CNN models in the Yeast dataset. Results of the KAML model from (Yin et al., 2020) are included for comparison.

This page intentionally left blank.

Chapter 7

Graphical methods

7.1 Introduction

When dealing with finite size populations, and without random mating, the *iid* assumptions of classical genetics regarding SNPs and individuals certainly will not hold. In the case of SNPs, correlations between markers will exist, a phenomenon that, as we already mentioned, is called *linkage disequilibrium*. Structure may also be found at a population level, due to kinship. Whether gains arising from exploiting population structure are desirable, or considered overfitting, depends on the ultimate goal of the analysis (Wray et al., 2013). It may be argued that for some agricultural applications testing samples are unlikely to deviate significantly from the training distribution since selection results in a highly structured population. Therefore, associations between both variables and samples are relevant in this context, and could be leveraged when making phenotype predictions.

Graphs are a common way to model and represent those relationships, and have thus been used extensively in genetics (Sinoquet, 2014). In particular, graphical models have been developed in the context of GWAS (Mourad, Sinoquet, & Leray, 2011) and genomic prediction (Morota et al., 2013).

On the other hand, the advent of Graph Neural Network architectures has enabled Deep Learning on non-Euclidean data (Wu et al., 2020a), establishing state-of-the-art results in many Graph Signal Processing tasks. They therefore present a promising research direction in the context of genome enabled prediction of complex traits which, to the best of our knowledge, has not been explored yet.

7.2 Basic definitions

Although great introductions in graphical models can be consulted (Kolaczyk & Csárdi, 2020), on this section we lay out (only) some necessary basic notions.

A graph or network $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ is defined as a set of nodes or vertices $\mathcal{V}$ which are connected by a set of edges $\mathcal{E}$. Edges are represented as a pair of vertices $\{u, v\}$ $u, v \in \mathcal{V}$, and an edge $\{u, v\}$ is said to be *incident* with the vertices u and v. If edges are symmetric, that is $\{u, v\} \in \mathcal{E}$ iff $\{v, u\} \in \mathcal{E}$, the graph is said to be

undirected. An example of such a graph is shown in Figure 7.1.

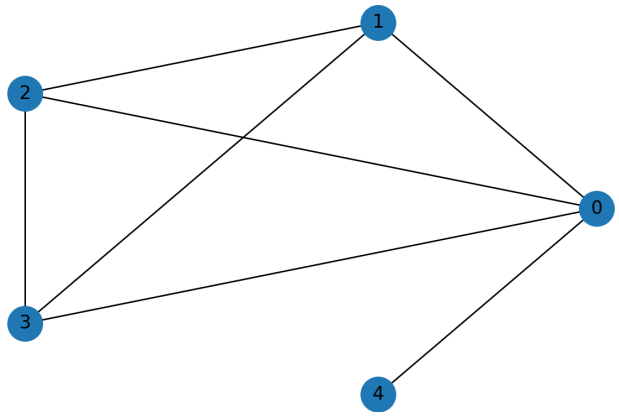


Figure 7.1: Example of an undirected graph. Nodes are represented by circles, which are labeled from zero to four. Edges between nodes are represented by lines. That is, a line is drawn between two nodes if there is an edge between them. Example from Zachary (1977).

Both vertices and edges can be labeled arbitrarily assigning integers $1, \ldots, |\mathcal{V}|$, and $1, \ldots, |\mathcal{E}|$ respectively. Graphs differing only on relabelings of its vertices and edges, are *isomorphic*.

A *subgraph* $\mathcal{S}$ is a graph made up of a subset of the original nodes and edges of the graph:

$$\mathcal{S} = (\mathcal{V}', \mathcal{E}') \text{ is a subgraph of } \mathcal{G} = (\mathcal{V}, \mathcal{E}) \text{ iff } \mathcal{V}' \subseteq \mathcal{V} \text{ and } \mathcal{E}' \subseteq \mathcal{E}.$$

An *induced subgraph* is a subgraph made up by a subset of the nodes and *all* the edges corresponding to those nodes:

$$\mathcal{S} = (\mathcal{V}', \mathcal{E}') \text{ is a subgraph of } \mathcal{G} = (\mathcal{V}, \mathcal{E}) \text{ iff } \mathcal{V}' \subseteq c \text{ and } \mathcal{E}' = \{\{u, v\} \in \mathcal{E} \ \forall \ u, v \ \in \mathcal{V}\}$$

A subgraph of the graph represented in 7.1 is shown on Figure 7.2.

The *adjacent nodes* of node u are the set of nodes that share an edge with u.

$$N_1(u) := \{v : (u, v) \in \mathcal{E}\}.$$

In an abuse of notation, the word *neighbourhood* or *one-hop neighbourhood* is sometimes used to describe the adjacent nodes and also the *subgraph induced* by those nodes.

A path P is a sequence of nodes $P = u_0, \ldots, u_k$, such that consecutive nodes are connected: $(u_i, u_{(i+1)}) \in \mathcal{E}$. The path is said to have length $|P| - 1$.

The k hop neighbourhood of node u is defined as the set of nodes for which a path with length k to u exists:

$$N_1(u) := \{v : (u, v) \in \mathcal{E} / \exists \bar{P_k}(u, ..., v) \text{with length k.}$$

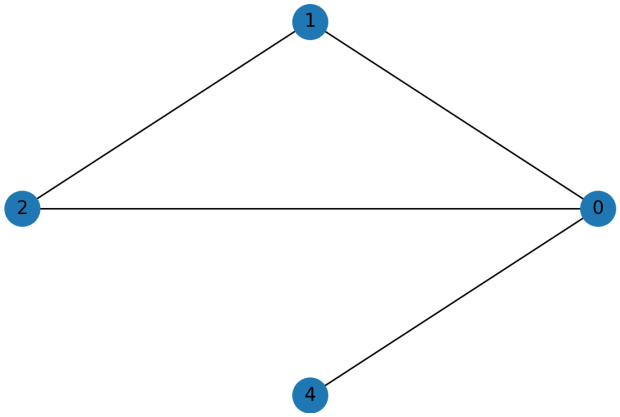


Figure 7.2: Example of the subgraph induced by nodes $\mathcal{V}' = 0, 1, 2, 4$ from the graph represented on Figure 7.1.

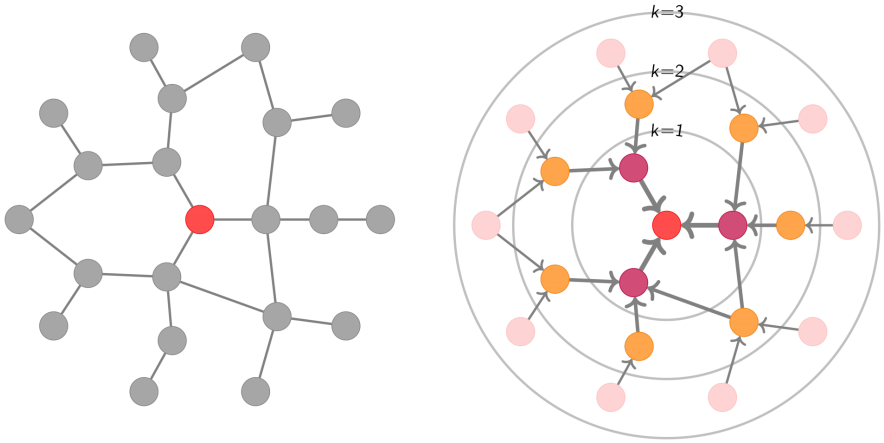


Figure 7.3: *k-hop* Neighbourhoods example. On the left, the graph with the target node coloured in red is shown. On the right, nodes belonging to the $k - hop$ neighbourhoods are coloured differently. Circles are also drawn to illustrate neighbourhoods. Figure by Gaudelet et al. (2020).

An undirected graph is *connected* if there is a path between any pair of nodes $u, v \in \mathcal{V}$.

Feature Vectors can be added to both nodes and edges. The terms *node features*, *signals supported on graphs*, or *graph signals* are all used to refer to mappings $x : \mathcal{V} \to \mathbb{R}^n$, where n is the dimension of the feature space. Figure 7.4 is Edge

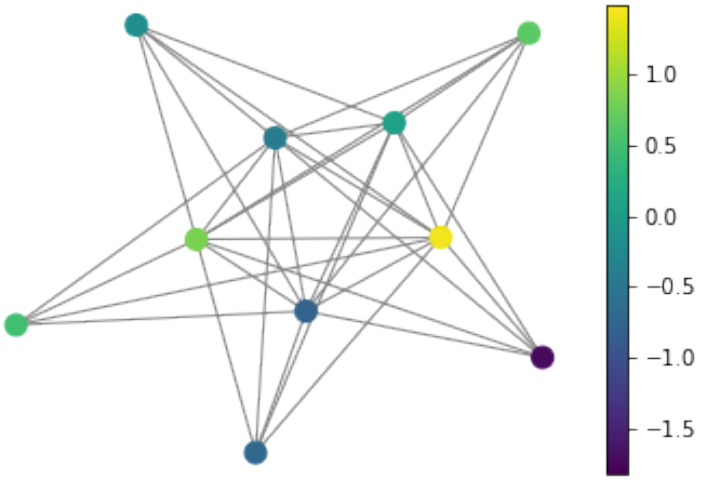


Figure 7.4: Example of a graph supported signal. Each node (circles) represents a Holstein animal, and the signal is a real valued phenotype (color). Edges (lines) represent correlations between genotypes.

features are mappings $e : \mathcal{E} \to \mathbb{R}^m$. It is common to associate a scalar value with each edge, which is referred to as *edge weight*.

For a weighted graph to be *undirected*, these weights have to be symmetrical:

$$w_{ij} = w_{ji} \text{ for all } (i, j) \in \mathcal{E}$$

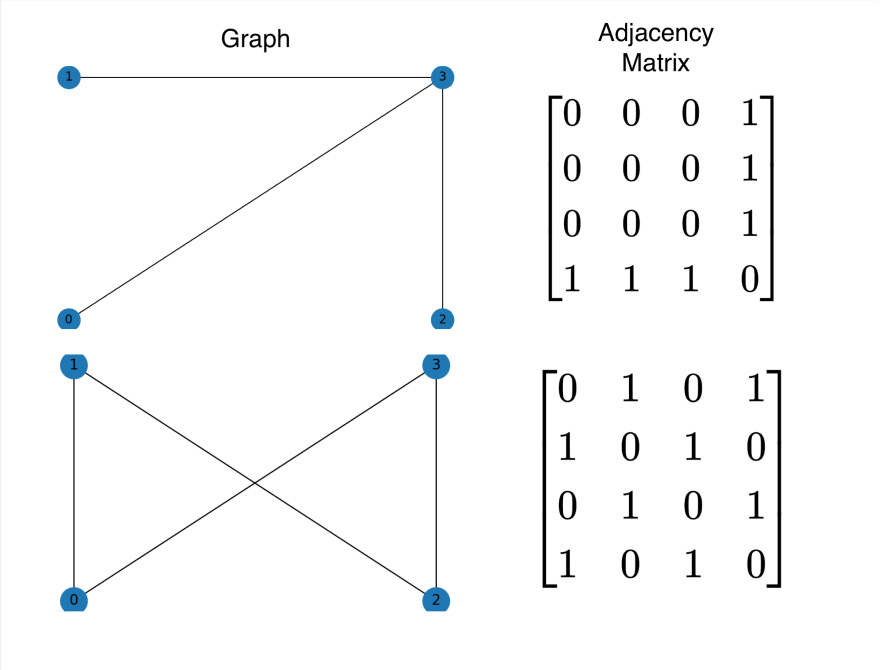
The adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, is a common representation of the graph, which is defined as:

$$A_{ij} := \begin{cases} w_{ij}, & \text{if } (i, j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases},$$

where w_{ij} is the weight assigned to the corresponding edge. Unweighted graphs can be seen as a special case of weighted graphs where $w_{ij} = 1 \forall\ (i, j) \in \mathcal{E}$. If the graph is undirected, its adjacency matrix will be symmetric.

A node's degree d_v is defined the sum of incident edge weights with node v:

$$d_i = \sum_j A_{i,j}.$$



$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 7.5: Examples of graphs and their adjacency matrices.

In the case of unweighted graphs, it is the number of adjacent nodes.

The degrees from all nodes of the graph are represented by the degree matrix $\mathbf{D} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$:

$$D_{ij} := \begin{cases} d_i, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

The normalized adjacency matrix is defined as:

$$\overline{A}_{ij} := \begin{cases} \frac{w_{ij}}{\sqrt{d_i d_j}} & \text{if } (i,j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases},$$

which can be expressed in matrix form as:

$$\overline{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$$

Another useful matrix is the graph Laplacian, $\mathbf{L} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

It can give a sense of smoothness of graph signals, which is commonly introduced by stating its connection with the Dirichlet Energy of a graph signal:

$$\mathbb{E}(\mathbf{x}) = \mathbf{x}^T\mathbf{L}\mathbf{x} = \sum_{\{i,j\}\in\mathcal{E}} w_{ij}\,(x_i - x_j)^2\,,$$

where $\mathbf{x}$ is a signal supported on a graph with Laplacian $\mathbf{L}$, and $\mathbb{E}(\mathbf{x})$ its Dirichlet Energy. The spectral analysis of the Laplacian matrix gives several insights about

a graph's structure and properties, as described by Chung and Graham (1997). Laplacian matrices can be normalized in the same way as adjacency matrices, defining the normalized Laplacian $\overline{\mathbf{L}}$:

$$\overline{\mathbf{L}} := \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$$

Normalized operators are thus closely related:

$$\overline{\mathbf{L}} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2} = \mathbf{I} - \overline{\mathbf{A}}$$

Both the Laplacian and the Adjacency matrix are Graph Shift Operators (GSOs), which are matrices $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ such that:

$$S_{ij} := 0, \text{ if } \{i, j\} \notin \mathcal{E} \text{ and } i \neq j$$

7.3 Graph Topology Inference

There are several ways to build a graph representation from SNP marker data. Individuals can be treated as nodes, and thus genotypes are treated as node features. Alternatively, SNPs may be treated as nodes, and thus each genotype represents a graph signal. Although several graphical models treating SNPs as nodes exist (Rosa, Felipe, & Peñagaricano, 2016), complex trait prediction literature regarding networks with individuals as nodes is scarce.

Kinship matrices can be used to construct population graphs, and biological information such as functional annotations, expression data and genetic distances can be used to build the SNPs-as-nodes graph (Kim et al., 2019; Lee & Lee, 2018; O'brien, Costin, & Miles, 2012; Thieffry, Huerta, Pérez-Rueda, & Collado-Vides, 1998; Tsalenko et al., 2006; L. Zhang & Kim, 2014). However, when this information is not available a graph representation may still be constructed from SNP marker data only, as an *association network* i.e., one where nodes with a sufficient level of *'association'* between node attributes are connected. Several measures of *association* can be employed, the most common being correlations and partial correlations.

In fact, the Genomic Relationship Matrix used in GBLUP (VanRaden, 2008) can be interpreted as the adjacency of an association network *between individuals*. Similarly, if an SNPs-as-nodes graph is constructed using the covariance matrix as a GSO, finding its spectral decomposition - which is called *Graph Fourier Transform* in the context of Graph signal Processing (introduced in Section 7.5.1) - is equivalent to performing PCA (Segarra, Huang, & Ribeiro, 2019).

It may also be noted that graph structures estimated from pairwise associations or distances are independent from node labeling. Another advantage is that, when performing inference, new nodes can be added without further information. This is crucial on population graphs, since test individuals are not part of the training graph. It could also be useful when treating SNPs as nodes, enabling the use of markers coming from different genotyping arrays.

Although treating SNPs as nodes and inferring network structure is biologically relevant since it may help uncover trait architecture and model complex gene

interactions, it was left for further work. When treating SNPs as nodes, the number of samples - that is, the number of individuals in the dataset - is significantly smaller than the number of parameters that need to be estimated ($p^2 \gg n$, where n is the number of individuals and p the number of SNPs). Some common regularisation techniques to overcome the ill-posedness of this problem are presented on Appendix D, but it remains challenging. Henceforth we will focus on the simpler approach, treating individuals as nodes.

Lastly, recent works in Graph Neural Networks have proposed network structure estimators that can be learned jointly with the task at hand (Kazi, Cosmo, Navab, & Bronstein, 2020; Y. Wang et al., 2019). This enables the use of GraphML methods on supervised tasks without prior knowledge about graph topology.

7.3.1 Association Networks

Measure of association or similarity.

The vast majority of *association network* literature focuses on continuous (and, more often than not, normally distributed) variables, while *SNPs are intrinsically categorical*. For haploid organisms, bi-allelic polimorphisms constitute binary variables and consequently some techniques may still be valid - namely phi coefficients are equivalent to pearson correlation coefficients derived for continuous variables. This is not the case for organisms with higher ploidy levels, where an appropiate *contingency analysis* should be carried out in order to infer association. However, treating the count of rare alleles at a given site as a continuous variable is a common practice on classical quantitative genetics, which is rooted on the extensive use of *additive* (linear) models.

When constructing association networks, partial correlations are often preferred, from the stance that edges should represent direct *associations* or direct effects between vertices. That is, when considering two variables the effect of the remaining variables should be adjusted. Under gaussianity assumptions, partial correlations are proportional to the inverse of the covariance matrix (**?**), known as the precision matrix. This is widely used in the context of *Gaussian Graphical Models* and several methods for estimating the precision matrix exist.

Conditional Independence Tests have been derived for non-gaussian variables (Belda, Vergara, Safont, & Salazar, 2019; Ramsey, 2014). Most are based on the idea of mapping the data to an appropriate reproducing kernel Hilbert space and measuring association on that space, i.e. they are kernel methods. These are clearly more suitable for genomic data, and although they were not employed in the current work, deriving graphs from these tests is a promising research direction.

In these thesis we have employed Pearson's Correlation coefficient, defined in Equation 2.5, as a measure of similarity between individuals. Exploring other association measures, as the ones mentioned above, is left for further work.

Sparsification

Inducing sparsity is also a crucial part of network estimation. The aim is to reduce spurious correlations that appear as a result of multiple testing, induce structure and stability on network estimators, and keep down costs in terms of memory and computation for subsequent processing algorithms, since the number of nodes can be large.

Figure 7.6 shows subgraphs consisting of 50 individuals chosen randomly from the Holstein dataset, obtained using different sparsification methods. In Appendix E the graph topology obtained using different sparsification methods and levels of sparsity is analysed. This analysis is centered on Holstein dataset, and the graphs which were then used to train and evaluate predictive models.

The two sparsification methods used are outlined below.

Thresholding This simple heuristic consists in directly setting to zero elements of the GSO matrix that fall bellow a certain threshold to attain a desired sparsity level. It has been shown that under certain conditions it is equivalent to Graphical Lasso (Fattahi & Sojoudi, 2019).

k-NN It consists of keeping only the top k Nearest Neighbours for each node, i.e. keeping the k larger correlations. This can be done either row-wise or column-wise. Note that the resulting GSO will not be symmetric, and therefore the resulting graph will be directed. We have chosen to set k neighbours as incident edges, and as a result the obtained graphs will have a fixed *in-degree*.

7.3.2 Random Graphs

In order to assess the effect of graph topology on predictive accuracy, we compared the performance of graphical models trained on association networks with models trained on random graphs. For this purpose, two types of random graphs were explored.

Erdös–Rényi model

In the Erdös–Rényi random graph model, graphs are built by randomly connecting nodes. Precisely, the edge weights are modeled by independent Bernoulli random variables:

$$e_{ij} = \begin{cases} 1 & \text{with probability p} \\ 0 & \text{with probability 1-p} \end{cases} \qquad \forall i, j \in \mathcal{V} \quad i \neq j$$

Since these Bernoulli variables are independent, the distribution of degrees in the graph corresponds to a binomial distribution. The parameter p was chosen so that the expected mean degree of the graph equals 40 (which was the mean degree obtained using kNN and thresholding).

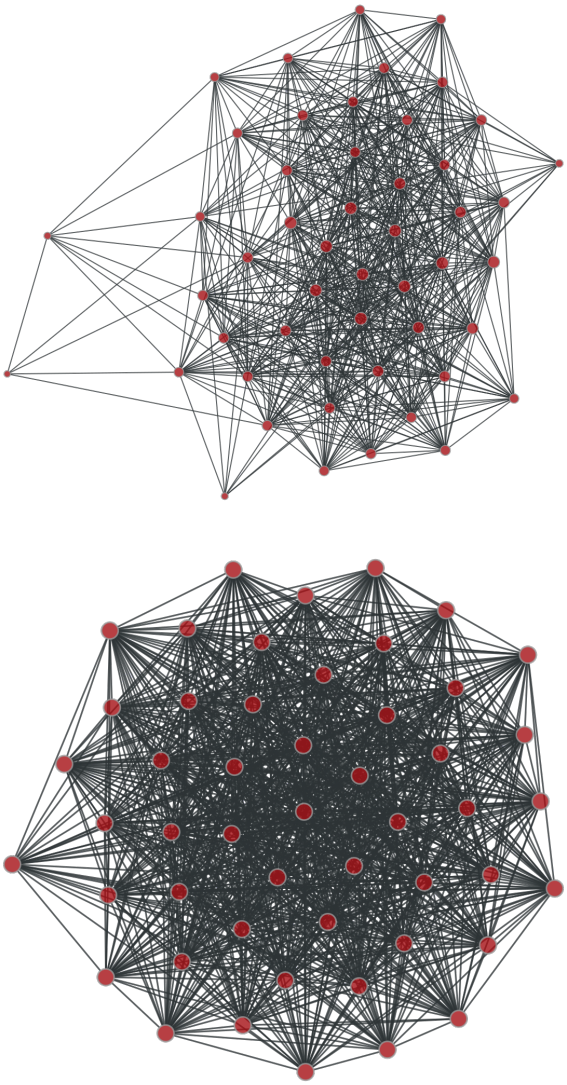


Figure 7.6: Graphs constructed using 50 individuals from the Holstein dataset. Pearson correlation and two different sparsification methods were used: thresholding at 0.55 (up) and 40-Nearest Neighbours (down). The size of nodes is proportional to their in-degree.

7.4 Problem formulation

7.4.1 Node Regression (Population Networks)

We will solve the risk minimization problem already stated in 2.1, restricting the function class $\mathcal{C}$ to graphical models:

$$\hat{\mathbf{y}} = \phi(\mathbf{X}, \mathcal{G}), \quad \phi \in \mathcal{C}$$

where $\mathbf{X} \in \mathbb{R}^{n \times p}$ is a matrix containing p-dimensional genotypes of n individuals, $\hat{\mathbf{y}} \in \mathbb{R}^{n \times m}$ are the m-dimensional predicted phenotypes of those individuals, and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph as defined on 7.2 with $|\mathcal{V}| = n$ nodes. A one-to-one mapping from nodes to individuals will exist, that is, each individual's genotype can be seen as node features in $\mathbb{R}^p$.

We only impose that ϕ depends on a graph, which is not given. Therefore, the choice of the graph $\mathcal{G}$ is also part of the learning problem. This is not true in many graph regression problems, where network structure is inherent, for instance if it has some physical meaning.

The problem with choosing an arbitrary graph is that models would not generalize when making predictions on unseen nodes which are not on the graph. It is therefore convenient to also find a function g that allows graph topology to be *inferred* from the data, that is $g(\mathbf{X}) = \mathcal{G}$. Suitable functions have been explored in 7.3.

In order to train, validate and evaluate models, random splits of the dataset will be used. At test time, training individuals may be involved (depending on the particular g chosen) in the inference of graph topology and phenotypes for test individuals. This is not a problem, even though re-computing the topology of the whole graph may be costly. However, it is important from a methodological perspective that test data is not used to construct the training graph, since we would incur in *data contamination*.

In our approach, predictions will only depend on the feature vectors of individuals and not their target outputs. Nonetheless, using phenotypes from other nodes on the graph would also be valid as long as no test phenotypes are used to predict other test phenotypes.

7.5 Graph Neural Networks

7.5.1 Introduction

In this section we explain the core components of Graph Neural Networks (GNNs).

Graph diffusions

Consider a graph $\mathcal{G}$ with N nodes and its adjacency matrix $\mathbf{A} \in R^{N \times N}$. Let $\mathbf{x} \in \mathbb{R}^N$ be a signal supported on graph $\mathcal{G}$, such that each x_i is associated to a single node in the graph. The dot product between the adjacency matrix $\mathbf{A}$ and

the graph signal $\mathbf{x}$ yields a *diffused* version of the signal. The diffusion corresponds to the weighted sum of the features of 1-hop neighbor nodes, where the weights correspond to edge weights. The following is an example of this process in an unweighted graph

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \quad \mathbf{Ax} = \begin{bmatrix} 6 \\ 2 \\ 1 \\ 9 \\ 8 \\ 7 \end{bmatrix}$$

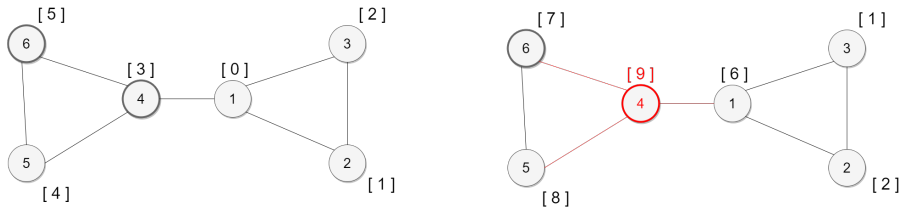


Figure 7.7: Example of the effects of a 1-hop graph diffusion on node 3.

The fourth row of $\mathbf{Ax}$ corresponds to the sum of node features connected to node 3. Note node 3's own feature is not included in the sum. In order to incorporate the central node in this aggregation process, self-loops are inserted into $\mathcal{G}$. This corresponds to taking the adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbb{I}$.

The result of the 1-hop diffusion is yet another graph signal $\mathbf{x}^{(1)}$, in which the value of each node (feature) $x_i^{(1)}$ has been combined with that of its 1-hop neighbouring nodes. Thus, we say that multiplying a graph signal by a GSO diffuses the signal across the graph.

This process can be repeated with the signal $\mathbf{x}^{(1)}$, yielding the signal $\mathbf{x}^{(2)} = A\mathbf{x}^{(1)}$. Node $x_i^{(2)}$ combines information from node $x_i^{(1)}$'s 1-hop neighbours (and itself), which in turn combine information from x_i's 2-hop neighbors. This procedure can be extended k times, yielding the signal $\mathbf{x}^{(k)} = A^k\mathbf{x}$, in which each node $x_i^{(k)}$ is a combination of node x_i and its k-hop neighbours.

Graph Convolutional Filter

Graph Convolutional Filters (GCF) are a tool for the linear processing of graph signals. Given a GSO $\mathbf{S} \in \mathbb{R}^{N \times N}$ and a set of coefficients $h_k \in \mathbb{R}$, a GCF $\mathbf{H}(\mathbf{S})$ is a polynomial on $\mathbf{S}$ such that

$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K-1} h_k \mathbf{S}^k$$

which corresponds to a weighted sum of different powers of the adjacency matrix, with K being the order of the filter. Thus, the filter is represented by a matrix

whose dimensions are the same as the GSO $\mathbf{S}$. The convolution between the graph filter $\mathbf{H}(\mathbf{S})$ and the graph signal $\mathbf{x}$ is defined as

$$\mathbf{h} \star_{\mathbf{s}} \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$$

The sum of powers of the GSO implies that GCFs aggregate information from local neighborhoods in the graph. The extent of the neighborhood is determined by the filter order K.

Multiple-Input-Multiple-Output (MIMO) graph filters enable the processing of multiple feature -or matrix- graph signals. In MIMO graph filters, scalars h_k are replaced by coefficient matrices $\mathbf{H_k}$, which act as a collection of filter banks. A convolution between a MIMO graph filter $\mathbf{H}$ and a matrix graph signal $\mathbf{X}$ is described by the following equation:

$$\mathbf{H} \star_{\mathbf{s}} \mathbf{X} = \sum_{k=0}^{K-1} \mathbf{S}^k \, \mathbf{X} \, \mathbf{H}_k$$

On a side note, the usual 1-D time convolutions can be recovered as a particular case of graph convolution, with a particular directed graph as illustrated on Figure 7.8.

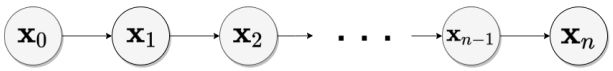


Figure 7.8: Time sequences as a directed graph.

Consider the case of a $n = 3$ (4-dimensional input signal). The adjacency matrix of the time series (defined as a graph) on Figure 7.8 is

$$\mathbf{S} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Thus

$$\mathbf{S}^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; \mathbf{S}^1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \; \mathbf{S}^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and considering the graph filter $\mathbf{h} = [h_0, h_1, h_2]^{\mathrm{T}}$, we have that

$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{2} h_k \mathbf{S}^k = \begin{bmatrix} h_0 & h_1 & h_2 & 0 \\ 0 & h_0 & h_1 & h_2 \\ 0 & 0 & h_0 & h_1 \\ 0 & 0 & 0 & h_0 \end{bmatrix}$$

Consequently

$$\mathbf{y}^{\text{GCF}} = \mathbf{H}(\mathbf{S})\mathbf{x} = \begin{bmatrix} h_0 x_0 + h_1 x_1 + h_2 x_2 \\ h_0 x_1 + h_1 x_2 + h_2 x_3 \\ h_0 x_2 + h_1 x_3 \\ h_0 x_3 \end{bmatrix}$$

Now consider a 1D convolutional filter $\mathbf{w} \in \mathbb{R}^3 : \mathbf{w} = [w_0, w_1, w_2]^{\text{T}}$, and the same signal $\mathbf{x}$ (as a time series) with an asymmetrical right zero-padding of width 2

$$\mathbf{x}_{\text{pad}} = [x_0, x_1, x_2, x_3, 0, 0]$$

The convolution between the filter and the signal $\mathbf{x}_{\text{pad}}$ is

$$\mathbf{y}^{\text{CONV}} = \mathbf{x}_{\text{pad}} \circledast \mathbf{w} = \begin{bmatrix} w_0 x_0 + w_1 x_1 + w_2 x_2 \\ w_0 x_1 + w_1 x_2 + w_2 x_3 \\ w_0 x_2 + w_1 x_3 \\ w_0 x_3 \end{bmatrix}$$

Thus, by setting $h_k = w_k$, $\forall k \in \{0, 1, 2\}$ we can recover the usual one-dimensional convolution. This simple 4-dimensional example can be easily generalized to sequences of arbitrary size.

GCFs in the frequency domain

GCFs can also be studied in the frequency domain (Ribeiro, 2020a). Consider a graph $\mathcal{G}$ with N nodes, described by its GSO $\mathbf{S}$. Let $\mathbf{V} = [\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{N-1}]$ and $\mathbf{\Lambda} = \text{diag}([\lambda_0; \ldots; \lambda_N])$ be the eigenvector and eigenvalue matrix of $\mathbf{S}$ respectively, such that $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{N-1}$. The matrix $\mathbf{S}$ can be expressed as

$$\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\text{H}}$$

The RHS of the above equation is often referred to as the *eigendecomposition* of the matrix. For undirected graphs, the GSO is symmetric, that is $\mathbf{S} = \mathbf{S}^H$, which implies $\lambda_n \in \mathbb{R}$, $\forall n$.

Using the eigendecomposition of $\mathbf{S}$, we define the Graph Fourier Transform (GFT) of graph signal $\mathbf{x}$ as

$$\tilde{\mathbf{x}} = \mathbf{V}^{\text{H}}\mathbf{x}$$

which corresponds to a projection of signal $\mathbf{x}$ to the eigenspace of the GSO $\mathbf{S}$.

Using this definition, we can define the GFT of a GCF. Recall the definition of a filtered graph signal

$$\mathbf{y} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$$

Substituting $\mathbf{S}$ by its eigendecomposition

$$\mathbf{y} = \sum_{k=0}^{K-1} h_k \mathbf{V}\mathbf{\Lambda}^k\mathbf{V}^{\text{H}}\mathbf{x}$$

And multiplying each side of the equation by $\mathbf{V}^{\mathrm{H}}$, we have

$$\begin{aligned}\tilde{\mathbf{y}} &= \mathbf{V}^{\mathrm{H}}\mathbf{y} \\ &= \mathbf{V}^{\mathrm{H}} \sum_{k=0}^{K-1} h_k \mathbf{V}\boldsymbol{\Lambda}^l \mathbf{V}^{\mathrm{H}}\mathbf{x} \\ &= \mathbf{V}^{\mathrm{H}}\mathbf{V} \sum_{k=0}^{K-1} h_k \boldsymbol{\Lambda}^k \mathbf{V}^{\mathrm{H}}\mathbf{x} \\ &= \sum_{k=0}^{K-1} h_k \boldsymbol{\Lambda}^k \tilde{\mathbf{x}}\end{aligned}$$

Since the matrix $\boldsymbol{\Lambda}$ is diagonal, the GFT of $\mathbf{y}$ can be written as

$$\tilde{y}_i = \sum_{k=0}^{K-1} h_k \lambda_i^k \tilde{x}_i$$

Thus, the frequency response of a GCF with coefficients $\mathbf{h} = \{h_k\}_{k=0}^{K-1}$ is given by

$$H(\lambda) = \sum_{k=0}^{K-1} h_k \lambda^k$$

where the λ variable is analogous to the frequency f in the regular Fourier transform. An important result of this analysis is that the GFT of the filter is completely independent of the GSO. However, the GSO determines the eigenvalues in which the filter's GFT is instantiated. Consequently, GCFs can be easily transferred between different graphs, which will instantiate its frequency response in different eigenvalues. This can be visualized in Figure 7.9.

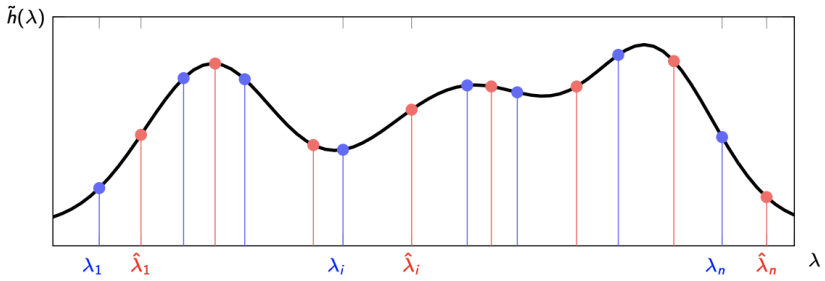


Figure 7.9: Example of a GCF's frequency response (black). The same filter is applied to two different graphs with different GSOs. As a result, each graph instantiates the frequency response in different sets of eigenvalues (denoted by the red and blue colors), defined by their respective GSOs. Image extracted from (Ribeiro, 2020a).

Graph Convolutional Neural Networks

As shown in Figure 7.10, Graph Perceptrons are non-linear maps consisting of a graph convolutional filter and a point-wise non linearity:

$$\Phi(\mathbf{x}, \mathbf{S}, h) = \sigma\left(\sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}\right)$$

where $\sigma(\cdot)$ denotes a non-linear function such as ReLU, $\mathbf{x}$ denotes a graph signal supported on a graph described by the graph shift operator $\mathbf{S}$ and h_k denote the convolutional filter coefficients.

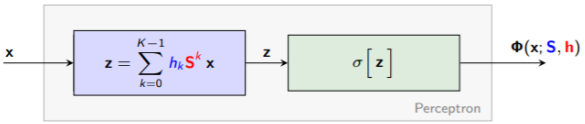


Figure 7.10: Diagram of a Graph Perceptron from (Ribeiro, 2020b)

In Graph Percepetrons, the trainable parameters are the filter coefficients h_k. Similarly, in MIMO Graph Perceptrons, the trainable parameters are the entries of the weight matrices of a MIMO graph filter:

$$\Phi(\mathbf{X}, \mathbf{S}, \mathbf{H}) = \sigma\left(\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X} \mathbf{H}_k\right)$$

where $\sigma(\cdot)$ denotes a non-linear function, $\mathbf{X}$ denotes a graph matrix signal supported on a graph described by the graph shift operator $\mathbf{S}$ and $\mathbf{H}_k$ denotes the weight matrices of the MIMO graph filter.

Graph Perceptrons are the building blocks of Graph Convolutional Networks, which achieve better generalization than graph filters (see Ribeiro (2020b)). Graph Convolutional Neural Networks are built by stacking Graph Perceptrons (see Figure 7.11). The trainable parameters of the 3-layer GNN shown in Figure 7.11 are weight matrices $\mathbf{H_1}$, $\mathbf{H_2}$ and $\mathbf{H_3}$. The layer-wise propagation rule for MIMO GCNs is the following:

$$\mathbf{X}^{\ell+1} = \sigma\left(\sum_{k=0}^{K-1} \mathbf{S}^k \, \mathbf{X}_\ell \, \mathbf{H}_{\ell k}\right)$$

One of the most popular implementations of Graph Convolutional Layers was introduced in Kipf and Welling (2016). This implementation corresponds to a first-order MIMO Graph Percepetron using a normalized adjacency with self-loops as the GSO and $\mathbf{H}_{\ell 0} = \mathbf{0}$. Its layer-wise propagation rule is the following:

$$\mathbf{X}^{\ell+1} = \sigma\left(\overline{\mathbf{A}} \mathbf{X}_\ell \mathbf{H}_\ell\right)$$

where $\overline{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{-\frac{1}{2}}$.

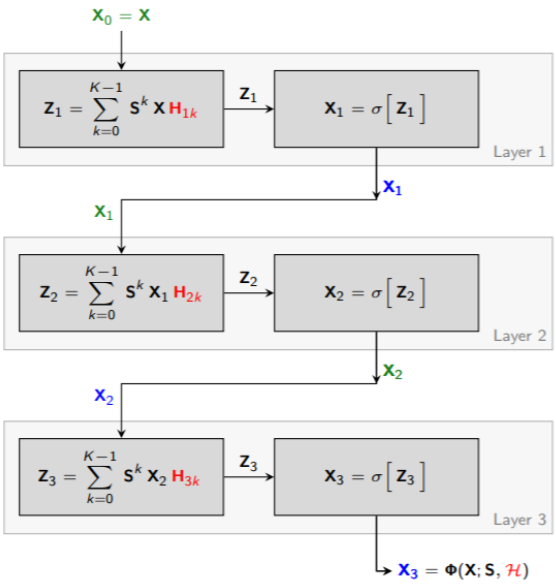


Figure 7.11: Diagram of a 3-layer MIMO Graph Neural Network from (Ribeiro, 2020b)

As explained by Gama, Isufi, Leus, and Ribeiro (2020), this implementation constrains the representation space of the GCNN and might be better suited for problems with small datasets.

7.5.2 Message passing Layers

As a result of the growing interest in Graph Neural Networks, several Architectures have been proposed (Wu et al., 2020b). *Message Passing* Neural Networks (MPNN) are popular high level framework, introduced by Gilmer, Schoenholz, Riley, Vinyals, and Dahl (2017), which encompasses many of these architectures. The basic building blocks of a MPNN are presented below.

MESSAGE: A "message" passing function that mediates the information exchange between a pair of nodes over an edge. In the case of a first order MIMO GCN, the message passing function is graph diffusion and a linear projection. That is, the message passing function between node i and j, with node signals $\mathbf{x_i}$, $\mathbf{x}_j \in \mathbb{R}^p$ is:

$$\text{message}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{S}_{j,i}\right) = S_{j,i}\mathbf{x}_j\mathbf{H}_1,$$

where $\mathbf{H}_1 \in \mathbb{R}^{p\times m}$ is the weight matrix of the MIMO graph filter with output dimension m, and $\mathbf{S}_{j,i}$ the edge weight between node j and i given by the GSO S.

AGGREGATE: An aggregation function that combines the collection of "messages" received by a node from its *1-hop Neighbours* into a single, fixed-length representation. These are usually permutation invariant functions.

In the first order GCN case it is simply the sum over all Neighbours:

$$\text{aggregate}\left(\{msg_{ji} : j \in \mathcal{N}(i)\}\right) = \sum_j msg_{ji},$$

where msg_{ji} are messages from node j to node i:

$$msg_{ji} = \text{message}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{S}_{j,i}\right).$$

UPDATE: An update function that produces node-level features given the previous features and the aggregated messages. This function usually includes a non-linear activation. In the GCN example it is the sum of aggregated messages and the node signal linear projection, followed by the pointwise non-linearity σ:

$$\text{update}(\text{agg}_i) = \sigma(\text{agg}_i + \mathbf{H}_0\mathbf{x}_i),$$

where $\mathbf{H}_0 \in \mathbb{R}^{p\times m}$ is the weight matrix of the MIMO graph filter with output dimension m, and agg_i are node i aggregated messages:

$$\text{agg}_i = \text{aggregate}\left(\{msg_{ji} : j \in \mathcal{N}(i)\}\right),$$

Then node i output at layer l are calculated as:

$$\begin{aligned} msg_{ji}^l &= \text{message}\left(\mathbf{x}_i^l, \mathbf{x}_j^l, \mathbf{S}_{j,i}\right) \\ x_i^{l+1} &= \text{update}\left(\text{aggregate}\left(\{msg_{ji}^l : j \in \mathcal{N}(i)\}\right)\right). \end{aligned}$$

Using the functions defined for the first order GCN, its nodewise Equation is:

$$\mathbf{x}_i^\ell = \sigma\left(\mathbf{H}_0^\ell\mathbf{x}_i^\ell + \sum_{j\in\mathcal{N}(i)} S_{j,i}\mathbf{H}_1^\ell\mathbf{x}_j^\ell\right) \tag{7.1}$$

The way in which this aggregation of local information is done (i.e: the choice of message, aggregate and update) is what distinguishes several types of GNNs.

Graph Attention

Introduced by Veličković et al. (2017), Graph Attention (GAT) uses attention scores when aggregating neighbourhood information.

The message function can be defined as:

$$\text{message}\left(\mathbf{x}_i^\ell, \mathbf{x}_j^\ell\right) = \mathbf{H}^\ell\mathbf{x}_j^\ell$$

where $\mathbf{x}_j^\ell \in \mathbb{R}^p$ are node j's features at layer l, and $\mathbf{H}^\ell \in \mathbb{R}^{p\times q}$ is a linear projection with output dimension q.

Then the aggregation function is a weighted sum, which gives weights α_{ij}^ℓ for each neighbour:

$$\text{aggregate}\left(\{\text{msg}_{ji}^\ell : j \in \mathcal{N}(i)\}\right) = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}^\ell \text{msg}_{ji}^\ell,$$

where the coefficients α_{ij}^{ℓ} are computed as the softmax of *attention scores* e_{ij}^{ℓ}:

$$\alpha_{ij}^{\ell} = \text{softmax}\left(e_{ij}^{\ell}\right) = \frac{\exp\left(e_{ij}^{\ell}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(e_{ik}^{\ell}\right)}$$

The function used to calculate e_{ij}^{ℓ} from node features $\mathbf{x}_j^{\ell}, \mathbf{x}_i^{\ell}$ is known as the *attention mechanism*, and several approaches exist (Weng, 2018). In *Graph Attention* (GAT) (Veličković et al., 2017) node feature embeddings $\mathbf{z}_i^{\ell}, \mathbf{z}_j^{\ell} \in \mathbb{R}^q$ are concatenated and fed to a single layer feedforward network:

$$e_{ij} = FCN\left(\mathbf{z}_i^{\ell} \| \mathbf{z}_j^{\ell}\right),$$

where FCN is a single layer perceptron with LeakyReLU activation, and $\|$ denotes concatenation.

Node feature embeddings $\mathbf{z}_i^{\ell}$ are also calculated using the linear projection $\mathbf{H}^{\ell}$:

$$\mathbf{z}_i^{\ell} = \mathbf{H}^{\ell} \mathbf{x}_i.$$

Note that this attention mechanism differs significantly from *dot product attention* which has been popularised in the Natural Language Processing domain by Vaswani et al. (2017).

The update function is a pointwise non linear function σ.

Edge Convolution

Graph edge convolutions, dubbed EdgeConv (Y. Wang et al., 2019), were originally proposed as a means to classify and segment point clouds. One of the main features that differentiates EdgeConv from other GNN approaches is that it dynamically recomputes the graph after each layer. As the nodes progress through the network, their embeddings mutate. Thus, proximity in the input feature space does not necessarily correspond with proximity in the intermediate layer's feature space. Updating the graph to reflect these changes in the feature space leads to nonlocal diffusion of information throughout the entire graph. In EdgeConv, the node features are aggregated after node embeddings have been calculated - and the pointwise non-linearity has been applied. This permutes the order of the nonlinear activation and aggregation against the previously described layers. That is, the aggregation function's output is the layers output. More specifically, the message function can be defined as:

$$msg_{ji} = \text{ReLU}(\mathbf{H}_{\phi}^{\mathbf{T}} \cdot \mathbf{x_i} + \mathbf{H}_{\theta}^{\mathbf{T}} \cdot (\mathbf{x_j} - \mathbf{x_i}))$$

where $\mathbf{H}_{\theta} \subseteq \mathbb{R}^{p \times M}$, and the parameter M is the number of channels of the node embedding function, i.e. the dimension of the layer's output embedding.

The max function is used as the channel-wise aggregator.

$$\text{agg}_i = \max_{j:j \in \mathcal{N}(i)} \text{msg}_{ji}$$

The update function in this case would be the identity:

$$update(\text{agg}_i) = \text{agg}_i$$

As mentioned earlier, after all new node embeddings are computed, the EdgeConv algorithm recalculates the graph on the new feature space. To do so, the graph samples the k-nearest neighbors of each node (using the same distance metric) and assigns them as the new neighbors, with k being a hyperparameter. This means that the network learns how to construct the graph instead of sticking to the original input graph's structure. As with other approaches, we utilize Pearson's correlation as the distance metric between nodes for the initial graph and $L-2$ dstance for intermediate graphs.

Readout Layer

In many architectures, a final layer called *readout layer* is added. The way in which the readout layer processes the graph signals depends on the downstream task. In node regression, the readout layer takes the node features of the last message passing layer and outputs predictions.

A common suitable choice for the readout layer in node regression, the output at node i can be computed as a simple affine transformation:

$$\text{output}_i\left(\mathbf{x}_i^{\ell'}\right) = \mathbf{W}^{\mathrm{T}}\mathbf{x}_i^{\ell'} + \mathbf{b},$$

where $\mathbf{x}_i^{\ell'} \in \mathbb{R}^{p'}$ are node features at the last layer ℓ'. $\mathbf{W} \in \mathbb{R}^{p' \times 1}$ and $\mathbf{b} \in \mathbb{R}$ are the weight and bias matrices, respectively, which can be trained jointly with the model.

In the context of genome-enabled prediction, initial node features correspond to genomes and the outputs of the readout layer are the predicted phenotypes. This can be easily extended to higher dimensional outputs, as in the case of multi-trait prediction.

Using 1D-CNNs

In all the graph neural network layers presented above, there is a step that consists of extracting embeddings from input node features. In the three of them, a linear projection is used for this purpose. Convolutional Neural Networks are a promising alternative for processing genomic data, as already stated in Chapter 6.

To see if performance gains could be achieved combining CNNs and GNNs, we used the residual CNN architecture described in Section 6.3.2.

Our objective was to use the CNN to calculate all node embeddings, *before* the neighbourhood aggregation step. However, since the CNN already had a considerable computational footprint, training a model that used the CNN to calculate the embeddings of *every node* in the neighbourhood was prohibitive in terms of memory and computational cost, even with moderate node degrees and batch sizes. To overcome this limitation we restricted the CNN to the target node's embedding,

and kept linear operators for neighbourhood operations. Both the CNN and the linear operators are *trained jointly* with the rest of the model.

Nonetheless, using shallower CNN models to extract features from neighbouring nodes also seems a promising research direction.

The following modifications to the above mentioned layers were introduced. First, we modified the GCN layer described in 7.3 to use a CNN in order to extract feature embeddings:

$$\mathbf{x}_i^{\ell+1} = \sigma\left(CNN(\mathbf{x}_i^{\ell}) + \sum_{j \in \mathcal{N}(i)} S_{j,i}\mathbf{H}_1^{\ell}\mathbf{x}_j^{\ell}\right) \tag{7.2}$$

As an alternative, we also concatenated the output embeddings of the GCN and the CNN blocks:

$$\mathbf{x}_i^{\ell+1} = \sigma\left(CNN(\mathbf{x}_i^{\ell}) \,||\, \sum_{j \in \mathcal{N}(i)} S_{j,i}\mathbf{H}_1^{\ell}\mathbf{x}_j^{\ell}\right) \tag{7.3}$$

where $||$ represents concatenating output vectors.

A high level overview of the architecture is shown in Figure 7.12.

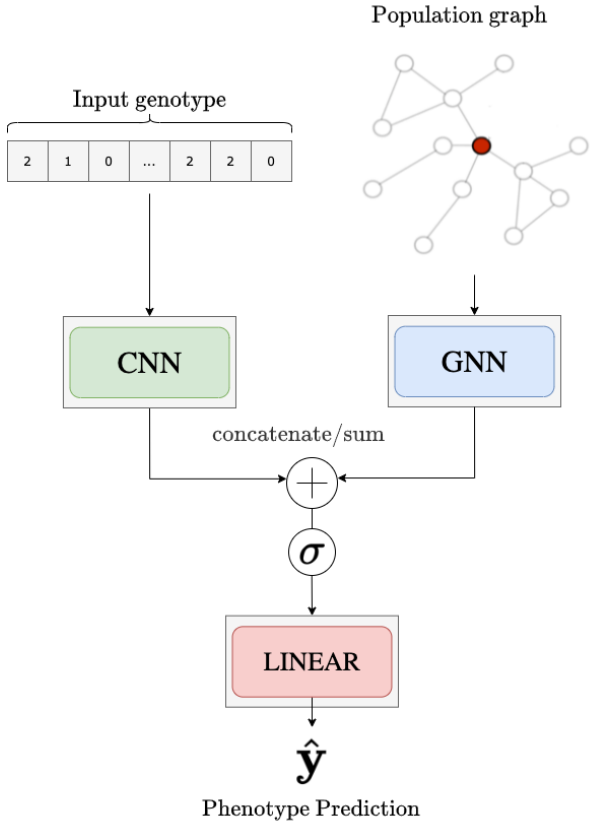


Figure 7.12: Mixed model, takes as inputs an individual as well as its neighbourhood in a population graph. The GNN block consists of a single GCN, Edge-Conv or GAT layer. The residual CNN constitutes the other block.

Experiments

All of the experiments and models described in this Section will be tested solely with the Holstein cattle dataset, which was chosen because it has the largest number of samples and markers. We also restricted the analysis to single trait prediction, and decided to focus on milk yield, which is a trait with a fair degree of complexity. How this analysis and results generalize to other traits and datasets requires further examination. We used 8 train-test data splits, and report results for all splits for each model and experiment, along with mean or median performance metrics.

Training procedure

The memory requirements for using the whole graph at each gradient descent step would be prohibitive. Besides, large batch training can lead to sharper minima which generalize poorly (Keskar et al., 2016).

Mini-batches are constructed by randomly sampling a number of target nodes and their k-hop neighbourhoods as shown in Algorithm 2.

Algorithm 2: Graph Node regression mini-batch SGD.

```
for epoch in 1, ..., n_epochs do
    for batch in 1, ..., n_batches do
        Sample m nodes without replacement
        for node in 1, ..., m do
            Sample k-hop neighbourhood
            Compute prediction for the target node
            Compute gradients
        end
        Gradient descent step
    end
end
```

Note that the number of nodes required at each training step would depend not only on batch size but also on the size of the node's k-hop neighborhoods. As a result, memory and computation costs would not be constant for every batch if the size of neighbourhoods varies from node to node, and may become prohibitive for nodes with large degree.

A common technique used to overcome this difficulty is to sample fixed size neighbourhoods for each node at each training step, which also has the added benefit of inducing regularisation (Hamilton, Ying, & Leskovec, 2017) - it can be seen as a form of dropout. However, full neighbourhoods may still be used when making predictions. Figure 7.13 shows an example of a sampled subgraph which would correspond to a training mini-batch of size 2, i.e. with two target nodes. For each target node, 5 neighbours were sampled from its *1-hop* neighbourhood.

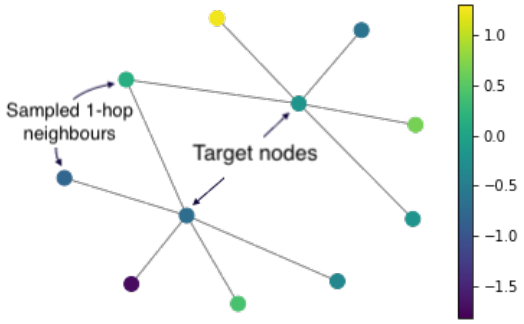


Figure 7.13: A 1-hop neighbourhood training mini-batch with 2 target nodes and 5 neighbours sampled for each target node. This mini-batch belongs to an association network constructed using the Holstein dataset, using pearson correlation between genotypes and thresholding at 0.55. Nodes are coloured according to the (scaled) Milk Yield value of the individual

7.6 Methodology

.

Each experiment type is ran on 8 different splits with 70% of the data reserved for training, 10% reserved for validation, and the remaining 20% for testing. The splits are seeded so that all experiments run on the same 8 splits. Models are trained with SGD with Nesterov Momentum, or the Adam optimizer when stated. The base learning rate for SGD with Nesterov momentum was 2.5^{-2} and 2.5^{-3} for Adam. A *Reduce On Plateau* callback was used to halve whenever training error stopped declining for five consecutive epochs. Models were trained for a maximum of 50 epochs with an early stopping callback, with a tolerance of 10 epochs to prevent overfitting. All weight matrices were intitialized via the Glorot Uniform initializer (Glorot & Bengio, 2010b), while biases were initialized to zero.

We conducted several experiments to evaluate different aspects about the graphical models and the underlying graphs:

- GNN Layers: first order GCN, EdgeConv, and Attention in Section . We explore one and two layer architectures. All models included batch normalization before the non-linear activation and dropout at the input.
- Approaches towards constructing the initial graph: different sparsification methods (kNN, thresholding), and different sparsification levels (i.e. thresholds and number of neighbors), in Section 7.7.2.
- Transferring GNNs trained on one graph to another, we also include random GSOs and a FCN as baselines, in Section 7.7.3.
- As in discussed in Section **??** we present saliency maps (7.7.4), activation histograms (7.7.5) and weight histograms (7.7.6), in order to further analyse

trained models.

- We compare MSE and Pearson correlation as Loss functions, as well as SAM and Adam as Optimizers, in Section 7.7.8.
- GNN and CNN embeddings aggregation: we explore Sum and Concatenation in Section 7.7.7.
- A Comparison against other models in Section 7.7.9.
- An evaluation on model size and performance in Section 7.7.10.

7.7 Results and Discussion

7.7.1 Architectures.

Graph Convolution, Edge convolution and Attention layers, as already described, were used to construct single and two layer models. In the two layer model, the second layer's inputs are the lower dimensional features or embeddings extracted by the first layer. This aims to aggregate relevant information, which may be more challenging in the high dimensional input space. Furthermore, repeated aggregation or diffusion can also enable the propagation of information to and from nodes further away. However, having more layers results in higher model capacity, which could also hinder generalisation.

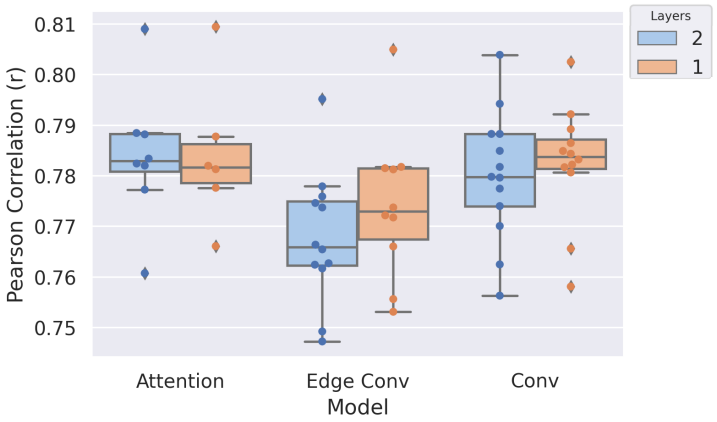


Figure 7.14: Single and two layer architectures' predictive performance trained minimising MSE with SAM.

As shown in 7.14, adding a second layer does not improve predictive correlation and even reduces it in the case of edge and graph convolutional networks. Although graph topology defines the neighbourhood of each node, on the attention layer the *weights* of each neighbouring node are calculated using node features. This could explain the fact that it outperforms models that rely on predefined weights

to propagate information. Therefore, assessing how models are affected by shift operator is necessary.

7.7.2 Graph Topology.

Since graph topology is inferred from the data, it varies significantly depending on the method and hyperparameters used, namely the threshold and number of neighbours. Therefore, we evaluated the Graph Convolutional model's performance when training it on different Graphs. As Figure 7.15 shows, the variation in predictive performance observed was limited. Notably, the two best performing graphs (kNN with $k = 40$ and thresholding with $treshold = 0.55$) have the same mean degree. Even though, as shown in previous chapters, their graph topology is substantially different, the performance on the downstream task was similar. This raises further questions about the role the graph plays within the trained model.

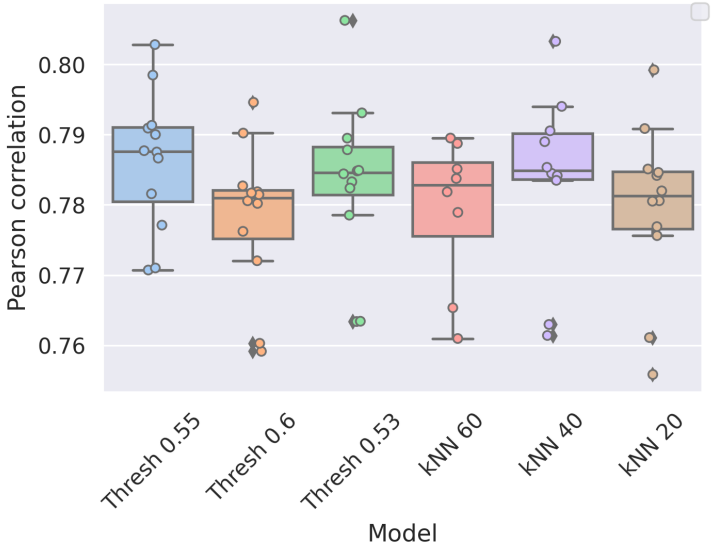


Figure 7.15: Graph Convolutional Network trained using different different Graph Shift Operators: k-Nearest Neighbours with $k = 20, 40, 60$ and thresholding with values $0.53, 0.55, 0.60$. The GNN is a single layer Graph Convolutional Network. Models are trained optimising r using *SAM*.

7.7.3 Transferability

All three architectures explored use different ways to agreggate neighbourhood information. In order to evaluate how graph topology affected the models, we employed kNN and thresholded correlation networks, as well as a random (Edrös-Rényi) adjacency matrix. All were set to have the same mean degree but, as already discussed, have significantly different topologies. The aim was to evaluate

how models trained on one graph performed when changing the underlying graph, i.e. the model's *transferabilty*.

As shown in Figure 7.16 model performance was not significantly affected by changing the underlying Graph Shift Operator. Furthermore, training a single layer FCN exactly like the one used to calculate target node embeddings in all three architectures performed comparably.

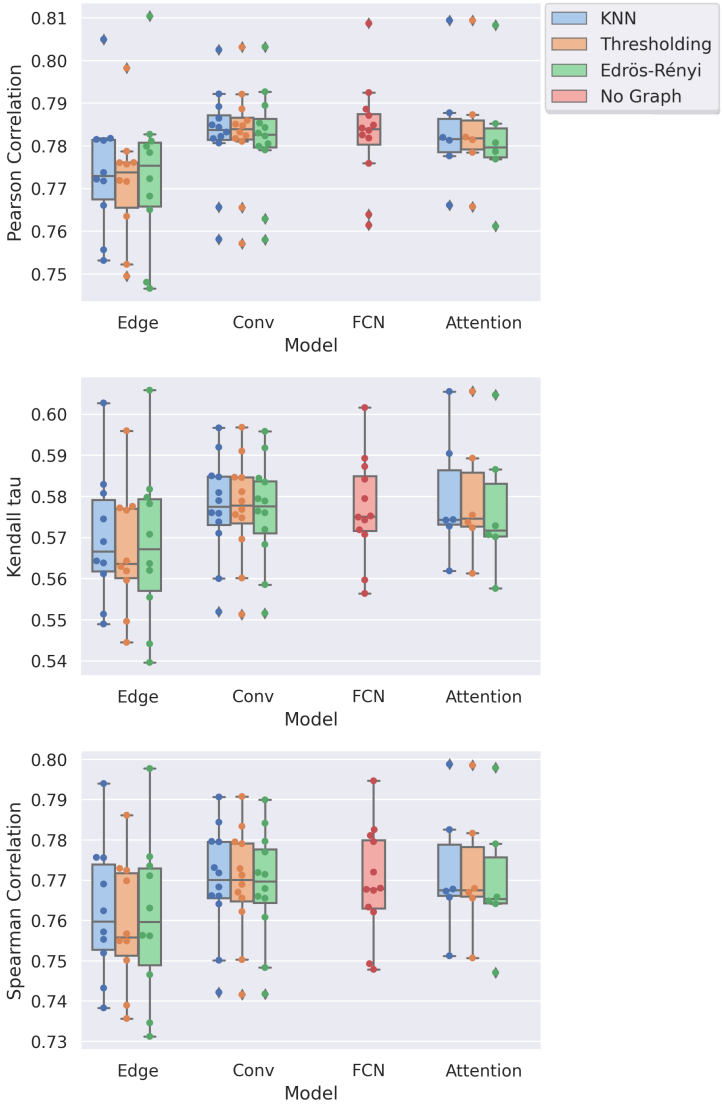


Figure 7.16: Performance metrics for different Graph Neural Network single Layer architectures, trained using a kNN graph and evaluated on kNN, thresholded correlation and Edrös-Rényi (random) GSOs. The same node predictor, which does not use any neighbourhood information (FCN) performs favourably.

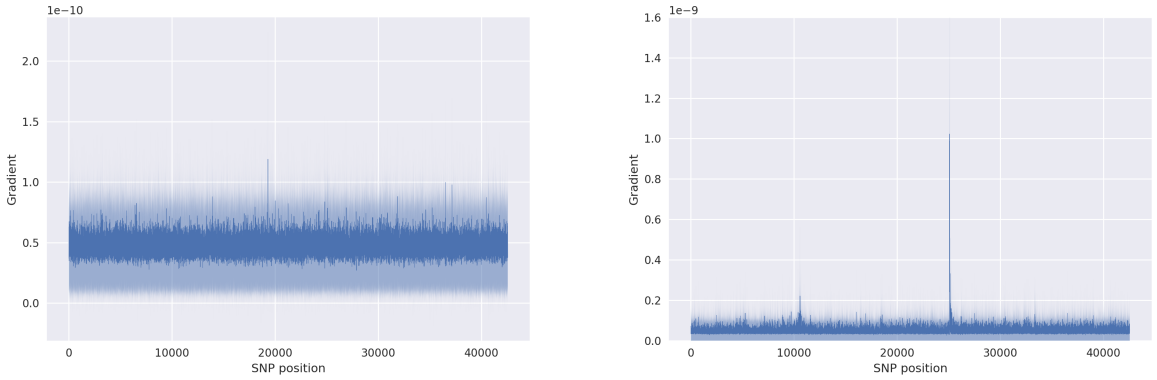
This result may suggest that the trained GNN models are exploiting node

feature only. Therefore, further analysis is required to assess how neighbourhoods affect predictions.

7.7.4 Saliency Maps

Figure 7.17 shows that the gradients of the GCN with respect to the target node are 10 times larger than the gradients with respect to the aggregated neighbourhood. This suggests that the target node has a bigger impact on the model's output than the neighbouring nodes, which is consistent with the transferability experiments.

The saliency map in Figure 7.17 (b) indicates that there are two regions in the Holstein genome which impact the model's output significantly. These regions are located in the proximity of SNPs 11000 and 26000, which is consistent with the saliency maps of the 1DCNN analysis**??**. The connection between the saliency maps obtained and the genetic architecture of this trait is discussed in Section 7.7.7.



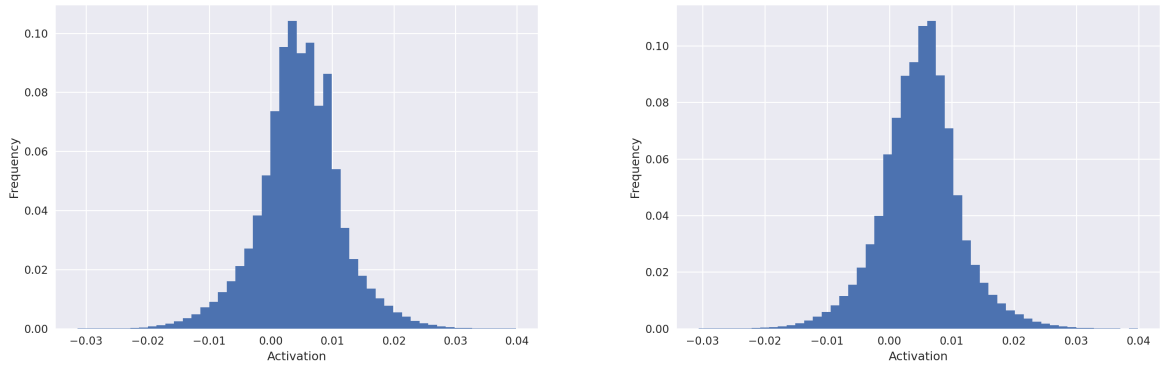
(a) Aggregated Neighbourhood

(b) Target nodes

Figure 7.17: Gradient magnitude with respect to the input signal at the aggregated neighbourhood and the target node. The mean of the gradients computed for all test samples corresponds to the solid line and its standard deviation to the shaded area.

7.7.5 Activation histograms

To further assess how neighbourhoods affect predictions, the distribution of activations from neighbouring and target nodes were computed. Consistently small neighbourhood activations would explain their lack of impact on the model's output. However, Figure 7.18 shows that the distributions of activations for the neighbouring and target nodes are similar. Both are gaussian-like distributions centered around 0.05.



(a) Neighbouring Nodes (b) Target Nodes

Figure 7.18: Histogram of target and neighbouring node activations for the test set.

7.7.6 Weights

Since the distribution of activations in the GNN layer does not shed light on the lack of impact of the neighbouring nodes, the weights of the linear predictor that follows the GNN layer were analyzed. Figure 7.19 indicates that the weights of the linear predictor associated to neighborhood embeddings are not significantly smaller than those associated to the target node embedding. Thus, the reason for the lack of impact of neighborhood embeddings on the output remains unclear.

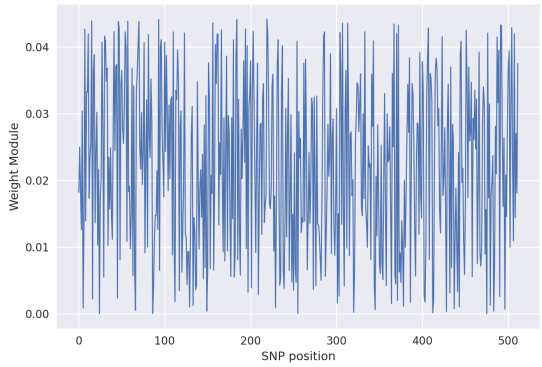


Figure 7.19: Weights of the linear predictor, neighbourhood embeddings correspond to the first 256 weights and target node embedding to the remaining 256.

7.7.7 GNN and CNN joint model.

Embedding aggregation

We proposed to combine CNN and GNN models, and train them jointly with the linear predictor. Two ways of aggregating the output of the GNN and CNN were tested. One is simply summing their outputs, and the other is concatenating those embeddings. This is then used as input to a linear predictor that outputs phenotypes. As shown on Figure 7.20, summing embeddings performed slightly better in terms of median predictive correlation and had a smaller interquartile range. When concatenating embeddings, the input dimenson of the linear predictor is doubled, and the decline in performance may stem from an increase in model capacity. That being said, both models perform comparably, and extending these analysis to other traits models and architectures calls for further research.

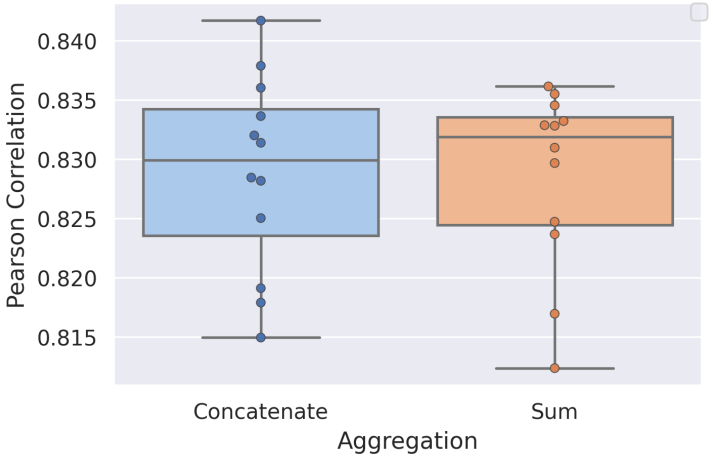


Figure 7.20: Graph Convolutional Network and GNN/CNN joint model, trained either summing or concatenating neighbourhood and node embeddings.

GNN architecture

All models showed an increase in predictive correlation when adding the CNN. However, as shown in Figure 7.22, only the GCN model outperformed slightly training solely the CNN. That is, when training both models jointly performs worse than when training the non graph model alone, in most cases. Whether training both models separately and then combining them as an ensemble has the same effect calls for further research. GNN architectures showed different gains in performance. Interestingly, the most complex and best performing GNN model (attention) showed little improvement when adding the CNN. In contrast a simpler (first order) GCN model showed a greater increase in performance.

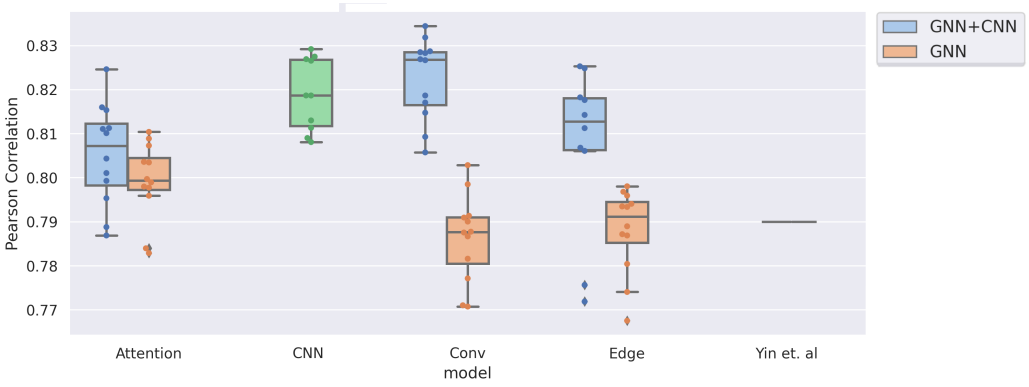


Figure 7.21: CNN only and CNN+GNN models trained using different GNN architectures. Results from (Yin et al., 2020) are included for comparison. kNN graph, and r as loss function were used.

Graph Topology

We assessed predictive performance when using different Graph Shift Operators to train and evaluate the the best GCN+CNN performing model. As Figure 7.22 shows, the choice of GSO does not have a significant impact on model performance. Therefore, evaluating the impact that neighbouring and target nodes have on final predictions is once again necessary.

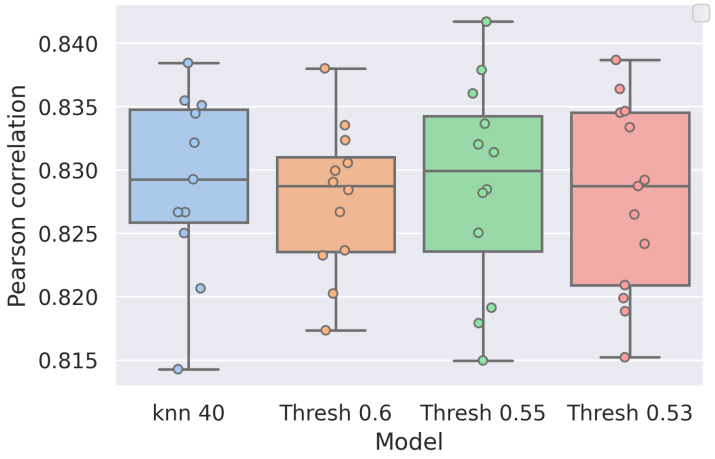
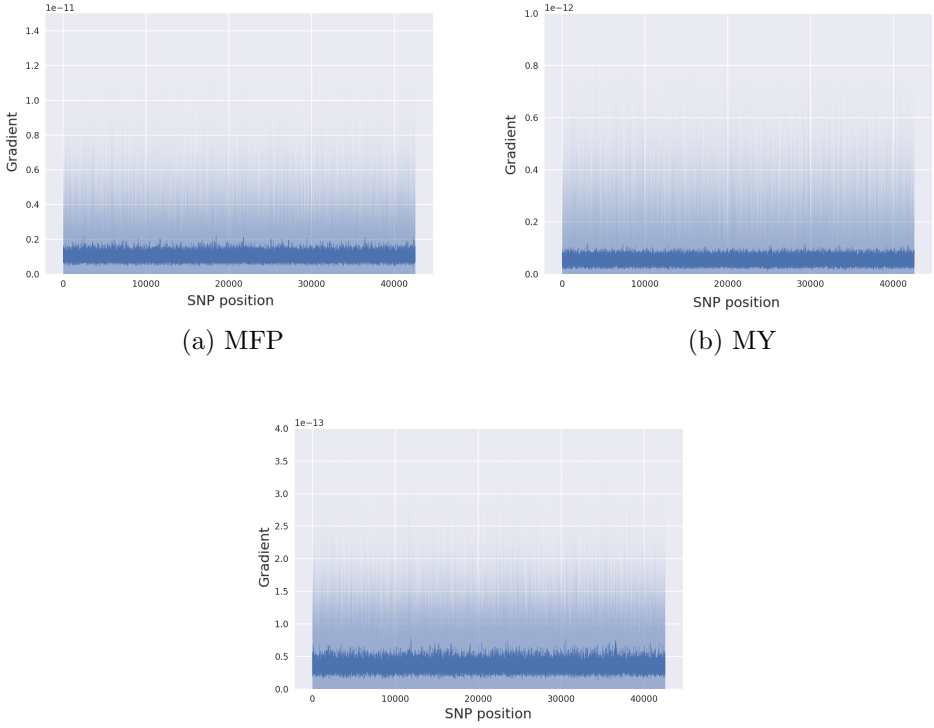


Figure 7.22: CNN+GNN models trained using different different Graph Shift Operators: k-Nearest Neighbours with $k = 40$ and thresholding with values 0.53, 0.55 and 0.60. The GNN is a single layer Graph Convolutional Network. Models are trained optimising r using *SAM*.

It is worth mentioning that results do vary to some extent depending on the GSO suggesting that neighbourhoods do play a role - albeit subtle. In that sense, thresholding correlation using a value of 0.55 proved to be the best GSO in terms

of predictive correlation.

Saliency Maps.



(a) MFP

(b) MY

(c) SCS

Figure 7.23: Mean gradient magnitude of the GNN with respect to the neighbourhood for calculated for test samples and all three Holstein traits.

Figure 7.23 shows the mean magnitude of the gradients of the output of the GNN with respect to the neighbours of an input sample. The fact that these gradients are nearly zero in all traits suggests that the model learns to ignore the neighbourhood of each node. This may explain why the predictive accuracy of the CNN and the CNN+GNN models are extremely similar.

Figures 7.24 (a,b,c) show similar saliency maps for all three models (CNN, GNN and Ridge). As already seen in Section 7.7.4, Figure 7.24 suggests that the three models largely base their predictions on two main sections of the Holstein genome: one around SNP 11000 and one around SNP 26000. The relative importance of these two sections varies between models, but in all of them, the region around SNP 26000 has the biggest impact on the model's output. The fact that contiguous SNPs tend to have a similar impact on the model's output is not surprising, since proximity is an indicator of correlation due to linkage disequilibrium, as shown in Figure B.2.

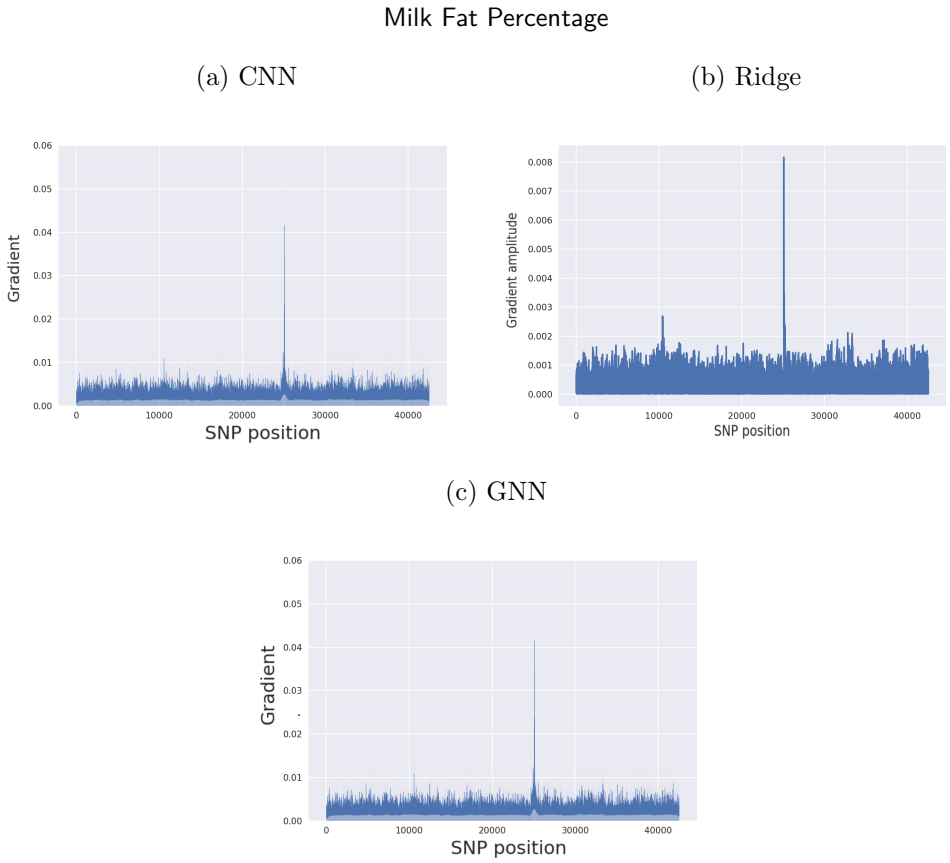


Figure 7.24: Gradients of the GNN+CNN, CNN and Ridge regression with respect to the input sample for Milk Fat Percentage.

Interestingly, the saliency maps in Milk Yield and Milk Fat Percentage are very similar. However, in MY the difference between the *impactful* regions (section around SNP 11000 and around SNP 26000) and the rest of the genome is more pronounced than in MFP. Moreover, there are subtle differences between the saliency maps of each model. As shown in Figure 7.25 (c) a section around SNP 33000 has a noticeable impact on the GNN's output. This is not the case for the CNN and Ridge. Similarly, in Ridge, a region around SNP 17000 has a high saliency map value. Both the CNN and the GNN attribute little importance to that region of the genome.

In contrast to the saliency maps of MFP and MY, Figure 7.26 indicates that, for all models, many SNPs have an impact on the model's output, and that these *impactful*[1] SNPs are spread out all along genome.

A key takeaway of this experiment is that, in all traits, the saliency maps of the three models (GNN, CNN and Ridge) are very similar. As mentioned earlier,

[1]Throughout this section we have deliberately avoided the term QTL, since its link with feature importance, LD, and genetic architecture requires a deeper discussion.

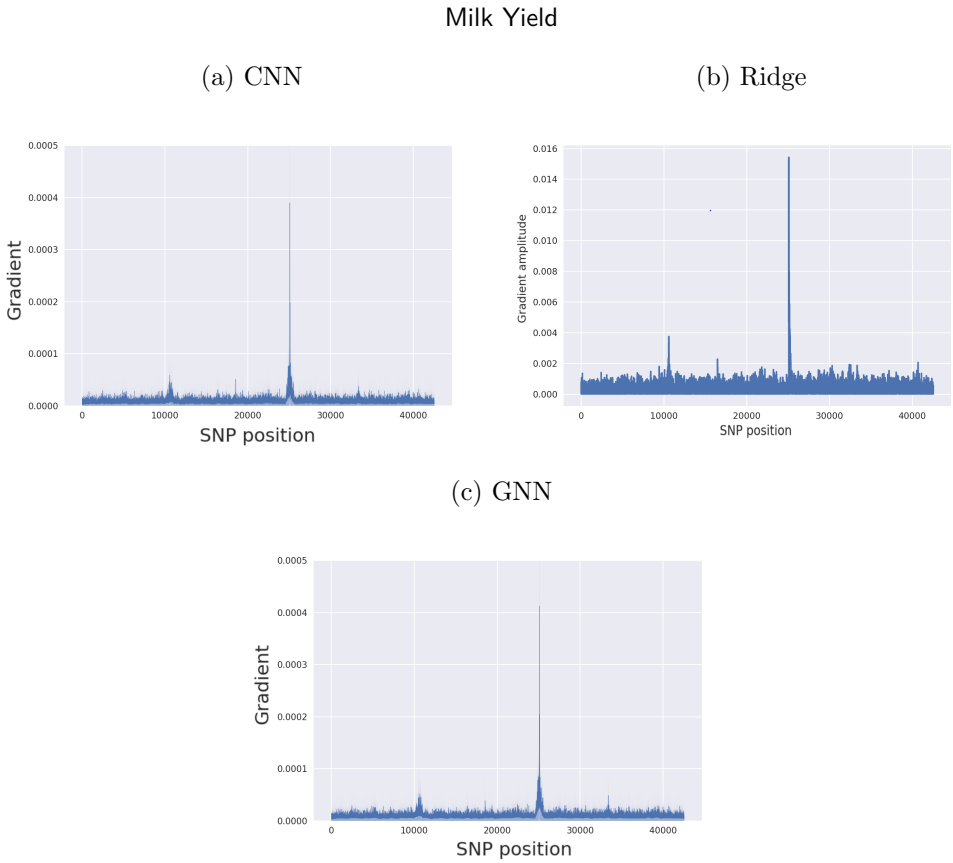


Figure 7.25: Gradients of the GNN, CNN and Ridge regression with respect to the input sample for Milk Yield.

saliency maps indicate the magnitude of the gradient of the output with respect to each feature of an input sample:

$$s = \nabla_x(f(x)).$$

Since a local first-order approximation of the model is: $f(x) \approx x \times \nabla_x(f(x))$, saliency maps can be viewed as the weights of a first order approximation of the model centered at point x. Therefore, the similarity between the saliency maps of the three models suggests that the first-order approximations of the three models are similar.

Secondly, the saliency maps obtained for MFP and SCS are consistent with the trait genetic architectures. Milk Fat Percentage has one or several major genes with large effect and many loci with small effect (Z. Zhang et al., 2015), and SCS has several loci with small or moderate effects. Both of these genetic architectures are reflected in the saliency maps computed. However, the genetic architecture of MY consists of a few moderate effect loci and many small effect loci. This is not evidenced in the saliency maps of Figure 7.25, where few input features have a very large effect.

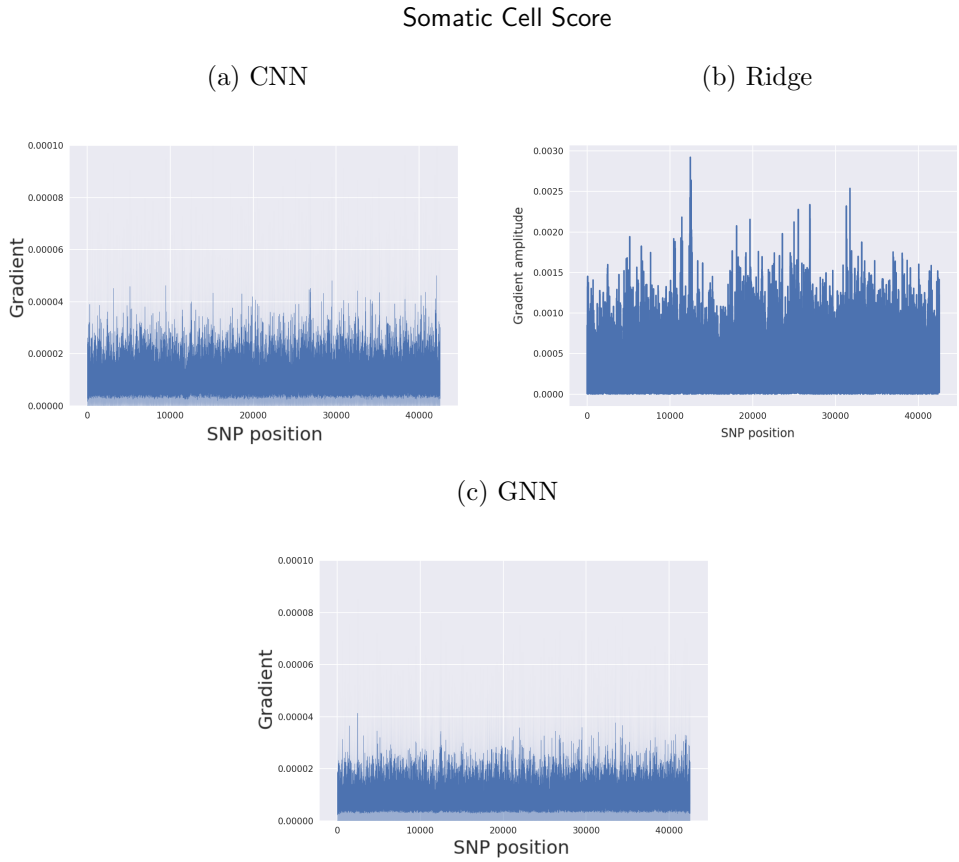
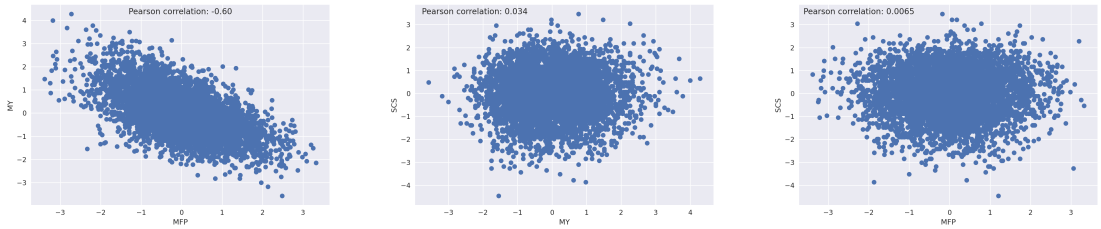


Figure 7.26: Gradients of the GNN, CNN and Ridge regression with respect to the input sample for Somatic Cell Score.

Finally, the saliency maps from MFP and MY are similar, but both very different from the SCS saliency maps. Similarly, MFP and MY phenotypes are highly correlated, while being uncorrelated with SCS, as shown in Figure 7.27. Similarity between the genetic architectures of the MY and MFP traits could explain both observations.



(a) MY-MFP (b) MY-SCS (c) SCS-MFP

Figure 7.27: Holstein phenotypes cloud points and correlation.

7.7.8 Optimization

- Optimizing r and then fitting a linear predictor, led to higher r, lower MSE and more stable training.
- Sharpness Aware Minimization showed a small improvement in performance for some models.

Loss function

In the context of genome-enabled prediction, the Pearson correlation coefficient is extensively used as a metric to compare models but not as a loss function in the ERM problem. A downside of using Negative Pearson Correlation ($-r$) as a loss function is its invariance to affine transformations, which may result in inaccurate predictions. As explained in Section 2.3, a model could maximize r by learning to predict a linear transformation of the targets: $\hat{y} = a \times y + b$, leading to a high MSE. Nonetheless, after the optimization, it would be possible to estimate the parameters a and b of this linear transformation. If the linear dependence between the observed and predicted phenotypes is high, undoing the estimated affine transformation should yield accurate phenotype predictions. To the best of our knowledge, this is the first approach to phenotype prediction that uses Negative Pearson Correlation as a loss function.

As expected, maximizing r led to higher predictive correlations than minimizing MSE (see Figure 7.28 (a)). However, when maximizing r, the $MSEs$ obtained are extremely high, as shown in Figure 7.28 (b). High correlation coefficients and high $MSEs$ suggest that the model predicts a linear transformation of the target values. We used the Least Squares method to estimate the parameters of such linear transformation:

$$\hat{a} = \frac{\sum_{i=1}^{N} (y_i - \bar{y}) \left(\hat{y}_i - \bar{\hat{y}}\right)}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$
$$\hat{b} = \bar{\hat{y}} - \hat{a}\bar{y}$$

where $y_1, y_2, ..., y_N$ are the target phenotypes and $\hat{y}_1, \hat{y}_2, ..., \hat{y}_N$ the predicted phenotypes. After undoing the estimated linear transformation, the predicted phenotypes obtained are: $y_i^* = \frac{\hat{y}_i - \hat{b}}{\hat{a}}$.

Surprisingly, undoing this linear transformation yields predictions with lower $MSEs$ than the model that uses MSE as a loss function (see Figure 7.28 (c)). We also applied this scheme to the model that uses MSE as a loss function. Reasonably, the linear dependence in this case was much weaker and the decrease in MSE was not significant.

The results presented above indicate that framing the genome-enabled prediction ERM problem as a Negative Correlation minimization problem can lead to higher correlations and lower mean squared errors. The cloud points before and after undoing the linear transformation in the CNN+GNN model are shown in Figure 7.29.

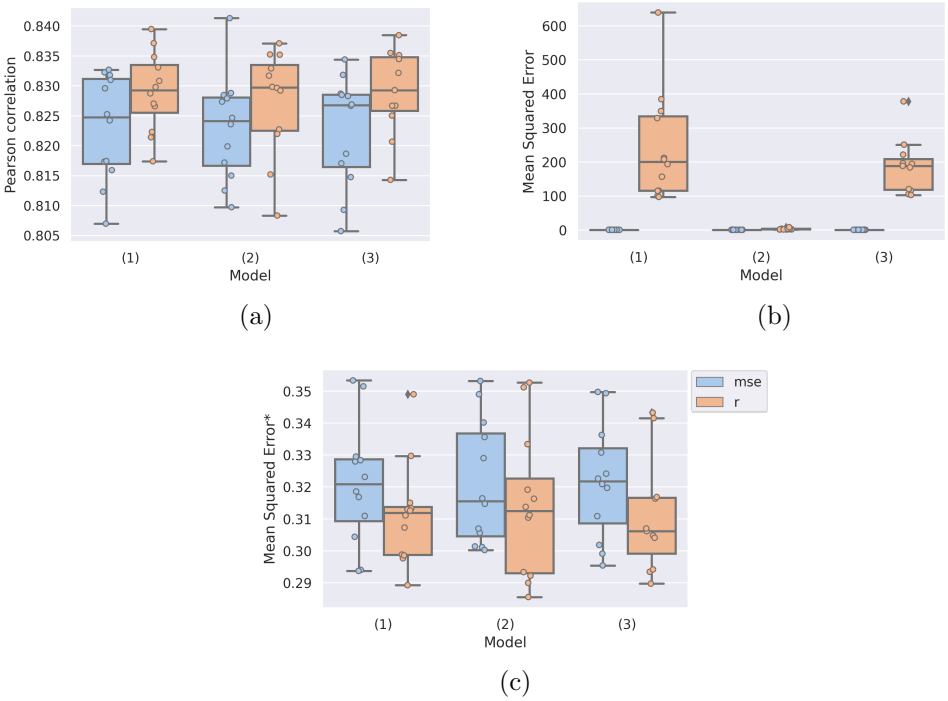


Figure 7.28: Pearson correlation, Mean Squared Error, and Mean Squared Error after undoing the estimated linear transformation, in the test set, with respect to the loss function optimized, for each of the following experiments: 1 - CNN+GNN optimized with SAM using thresholded GSO, 2 - CNN+GNN optimized with Adam using thresholded GSO, 3 - CNN+GNN optimized with SAM using kNN GSO

An increase in r may lead to a better ranking of the individuals. This is especially relevant in selective breeding, since selecting the top $X\%$ of individuals is common practice. In this case, predicting exact phenotypes or breeding values is not crucial, but getting an accurate ranking of the individuals is. In order to evaluate the extent to which an increase in r leads to a more accurate ranking, two popular metrics that assess similarity of orderings were used: Kendall's τ and Spearman's ρ.

One one hand, Kendall's τ is defined in terms of concordant and discordant pairs. A pair of observations $(\hat{y}_i, y_i)$, $(\hat{y}_j, y_j)$ is considered concordant if $\hat{y}_i - \hat{y}_j$ and $y_i - y_j$ have the same sign. In that case, concordance implies a correct ordering of samples i and j. A pair is discordant if it is not concordant.

$$\tau = \frac{(\text{ number of concordant pairs }) - (\text{ number of discordant pairs })}{\binom{N}{2}}$$

$$\tau = \frac{2}{N(N-1)} \sum_{i<j} \operatorname{sign}(\hat{y}_i - \hat{y}_j) \operatorname{sign}(y_i - y_j)$$

Figure 7.29: Predicted versus observed phenotypes with r (bottom) and MSE (top) as loss functions and the corrected -after undoing linear transform- predictions' MSE optimization.

On the other hand, Spearman's ρ is defined as the Pearson correlation coeffi-

cient of the rank vectors $r_{\hat{y}}$ and r_y. [2]

$$\rho_s = \frac{\mathrm{cov}\left(\mathrm{r}_{\hat{y}}, \mathrm{r}_y\right)}{\sigma_{\mathrm{r}_{\hat{y}}} \sigma_{\mathrm{r}_y}}$$

As shown in Figure 7.30, models that minimize the Negative Pearson Correlation are consistently better than models that minimize MSE in terms of both ranking metrics (Kendall's τ and Spearman's ρ).

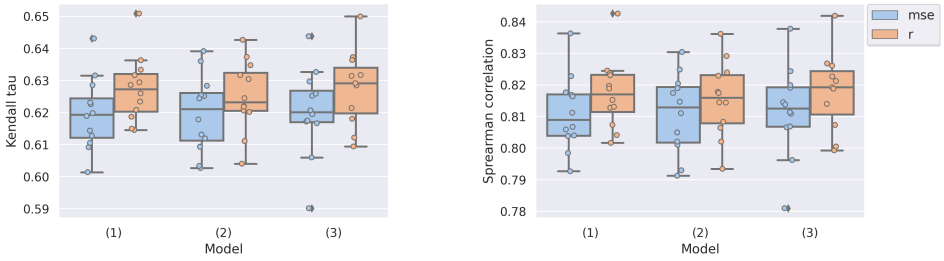


Figure 7.30: Loss function and ranking metrics, for each of the following experiments: 1 - CNN+GNN optimized with SAM using thresholded GSO, 2 - CNN+GNN optimized with Adam using thresholded GSO, 3 - CNN+GCN optimized with SAM using kNN GSO, 4 - CNN optimized with SAM

To further explore model performance with respect to rankings, a 2d histogram of predicted and ground truth rankings for one split of the CNN+GCN model is shown in Figure 7.31. Qualitatively, ranking improves slightly when optimizing r. In addition, rankings are most accurate at both ends, regardless of the metric optimised. Whether this observation generalises to other traits, datasets and models may be subject of further research. Accurately predicting which individuals have extreme values of a trait may be of practical value, and thus assessing prediction rankings may provide useful insights.

While useful for evaluating model performance, estimating and inverting the affine transformation using test data is not suitable in a real prediction setting and may lead to optimistic biases on reported metrics. However, the linear transformation can also be estimated using train data, and then applied to model predictions on inference. As Figure 7.32 illustrates, the performance of such approach in terms of MSE is similar to fitting the linear transformation using the test data.

This is important since simply fitting a linear predictor on top of models trained optimising r leads to both lower MSEs and higher predictive correlations (as well as the ordering metrics mentioned above).

[2] The i^{th} entry of a rank vector r_y holds the rank of the value in the i^{th} entry of the vector y.

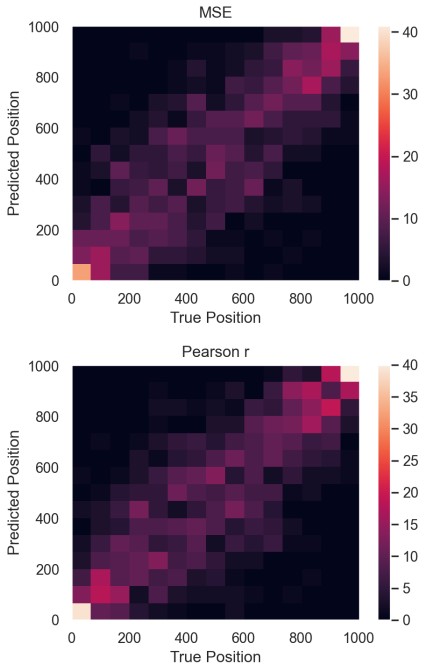


Figure 7.31: Ranking matrix of the sorted test target values when minimizing MSE and r.

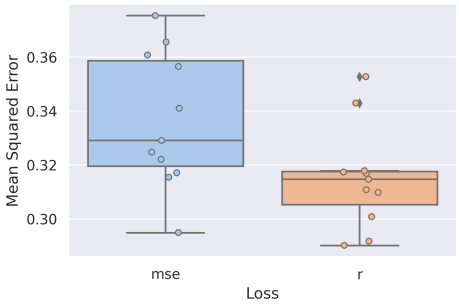
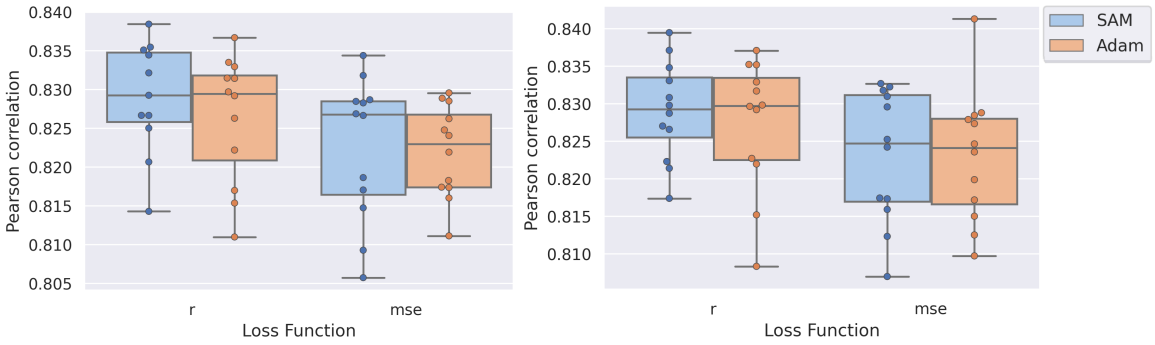


Figure 7.32: Loss function and Mean squared Error, when applying a linear regression fitted using train predictions. Once again, minimising r leads to lower MSE.

Optimizer

As shown in Figure 7.33 (a), when using a kNN GSO, the model optimized with SAM slightly outperforms its counterpart optimized with Adam regardless of the loss function used. When using a thresholded GSO, the average performance of Adam and SAM is similar in both loss functions. However, the models optimized with SAM and $-r$ as a loss function exhibit a slightly smaller interquartile range.

Figure 7.34 shows the evolution of the loss function and the validation r for the GNN+CNN model when being optimized for r and MSE. The top plot shows that



(a) kNN GSO (b) Thresholded GSO

Figure 7.33: Performance metrics with different loss functions and optimizers for the CNN+GCN model. (a) corresponds to the model trained using a kNN graph and (b) to the model trained using a thresholded GSO graph.

the loss curve is smoother when using $-r$ [3] rather than MSE as a loss function. This may be due to the fact that the loss is bounded, which may help prevent exploding gradients (Philipp, Song, & Carbonell, 2017), make training more stable and accelerate convergence. It could also be pointed out that correlation is more robust to outliers which could have larger loss values when using MSE.

Another important aspect about Figure 7.34 is that for both MSE and r, and especially regarding the latter, near zero training loss is attained. However, this does not hinder in validation or test performance, that is the model achieves good generalisation. This is known as the *interpolation regime* and although the generalisation power of over-parametrised models in this regime is not fully understood yet (S. Ma, Bassily, & Belkin, 2018), it is an active area of research.

[3] The loss plot shows $1 - r$ in order to facilitate the comparison.

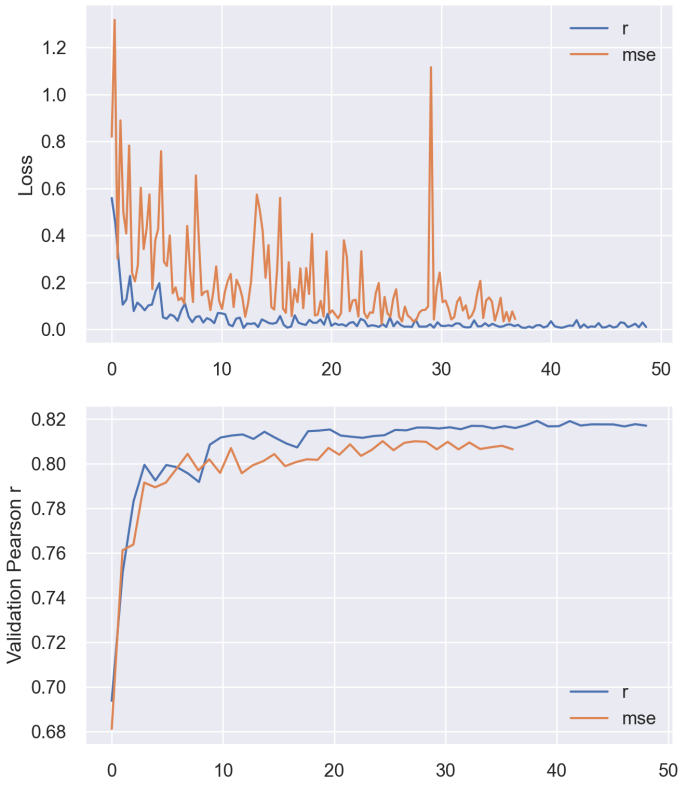


Figure 7.34: Loss curves for different loss functions.

7.7.9 Comparison against other methods

On Table 7.1 and Figure 7.35 the results of all models trained for Milk Yield prediction are included. For this particular dataset and trait, CNNs and CNN-GCN joint models show the best performance, when compared with other models proposed in the current work and the best results from literature (**?**). GNNs fail to outperform CNNs and literature, and CNN-GNN joint models fall below CNN performance for all but the GCN architecture, as already discussed.

7.7.10 Model Size

The number of model parameters is related to the model complexity. However, the relation between model size and generalisation has been recently challenged by deep learning and the emergence of *double descent* phenomena (Nakkiran et al., 2019). As illustrated in Figure 7.36, in some circumstances as model size increases

Table 7.1: Mean test Pearson correlation (r) for all models for the Milk Yield trait. Best results from literature - KAML (Yin et al., 2020) - are also included for comparison.

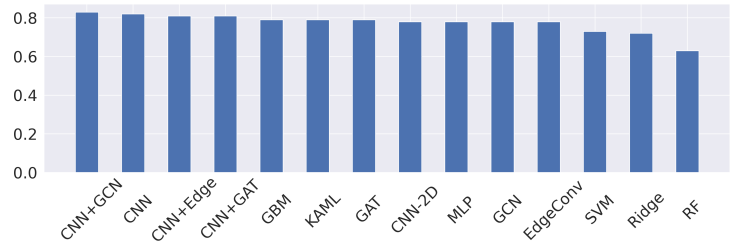


Figure 7.35: Mean test Pearson correlation (r) for all models for the Milk Yield trait.

beyond the classical U-shaped curve generalisation error falls again. This occurs at near zero training error, with highly overparametrised models.

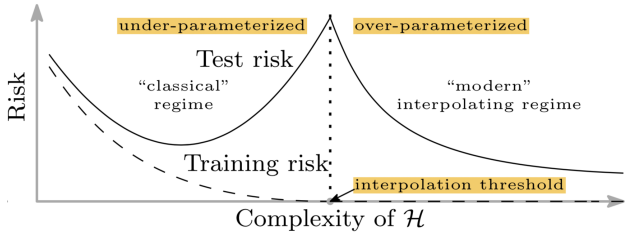


Figure 7.36: Double descent curve from (Nakkiran et al., 2019)

Furthermore, in the context of modern over-parametrized neural networks, it has been observed that, in some cases, pruning techniques can reduce the number of parameters by over 90 % without compromising predictive accuracy. Motivated by this observation, Frankl and Carbin (Frankle & Carbin, 2018) propose the lottery ticket hypothesis, which states the existence of subnetworks (called *winning tickets*) that can achieve comparable performance when trained in isolation. In this sense, the generalization power of over-parametrized neural networks may stem from the fact that the *effective* complexity of the model is small compared to the presumed complexity of the original neural network.

As shown in our best performing model for Holstein Milk Yield prediction had over two hundred million parameters, and was trained using only a few thousand samples. Whether the gains in performance outweigh the increase in model size and thus its training and inference computational cost, would depend on the specific use case or application. In that sense, the considerable increase in model size has only led to a small increase in performance, and increasing efficiency in this sense calls for further research.

Most *"Deep Learning"* models found in genome enabled complex trait prediction literature (Abdollahi-Arpanahi et al., 2020; Liu et al., 2019; W. Ma, Qiu, Song, Cheng, & Ma, 2017; W. Ma et al., 2018) actually have few layers and significantly fewer parameters than those presented in the current work.

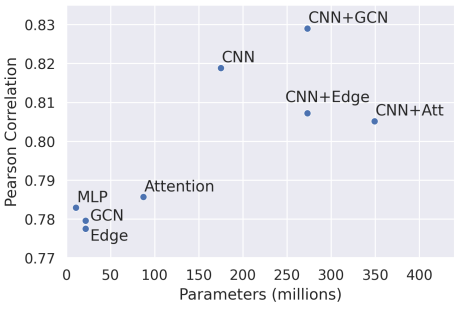


Figure 7.37: Selected models' test predictive correlation and number of parameters.

7.8 Summary

- All GNN architectures perform comparably to baseline models.
- Graph topology and neighbourhood information end up having little effect on predictions, even using random graphs.
- Training a joint GCN+CNN model results in a slight improve in performance, which surpasses the state of the art.

This page intentionally left blank.

Chapter 8

Conclusions and further work

As newcomers to the field of genome-enabled prediction, and genomics altogether, we find it interesting that a fully data driven approach could surpass state of the art results in most datasets. That is, by means of extensive hyperparameter tuning and nothing but Machine Learning *classical* methods we could obtain results that surpassed those found in literature. In that sense, tree ensemble and kernel methods performed favourably on most datasets but, unsurprisingly, no model achieved best performance consistently among all datasets and traits. It is also worth noting that these models outperformed bayesian linear regressions with carefully handcrafted and complex priors of marker association which draw upon domain knowledge.

Marker elimination ablation studies supported the well known fact that although genotypes are high-dimensional, numerous SNPs can be removed without affecting predictive power. This is due to the high correlation between SNPs owing to high linkage disequilibrium, because even if SNPs which have a large effect on the trait are removed, SNPs highly correlated with the removed ones may still be present. Although robustness to marker elimination thus depends on the genetic architecture of the trait and the structure of the population (through LD), rather surprisingly a similar behaviour was observed accross models and datasets. This result further motivates the study of dimensionality reduction techniques and representation learning in genomic prediction.

Another observation is that One-Hot-Encoding reduced model performance only slightly in most cases. This is of practical importance since OHE is one of the most popular ways of encoding categorical data for neural networks, and although the dimensionality of the input space increases, this did not hinder predictive ability significantly. The fact that performance did not improve either can be interpreted as a failure to capture non-linear or higher order effects, the *curse of dimensionality*, or the loss of one-to-one correspondence beween markers and variables. The extended use of additive encoding biases predictive models torwards additivity. This may be beneficial for many traits, but exploring other ways to encode SNP marker data which may favour more complex genetic architectures calls for further research.

In the light of the high dimensionality of input signals, and the fact that targets

are noisy since predictive power is limited by the trait's heritability, the data available on the datasets used is not abundant. Furthermore the architecture of all the traits studied had a strong linear component, as evidenced by the performance of linear models. One of the main focuses of this work was to explore the feasibility of employing modern deep learning architectures in this context. In this regard, it was possible to train highly over-parametrized architectures and still obtain good generalisation. For some datasets and traits, these models outperformed all others. However, this did not hold for all the models, traits and datasets studied. Besides, whether the gains in performance outweigh the increase in model size and thus its training and inference computational cost, and lack of interpretability, calls for further discussion.

In order to try to exploit local structure in the genome arising from linkage disequilibrium and cross-marker interaction phenomena, we explored the applicability of convolutional neural networks. Using residual connections improved performance across different traits and datasets. The fact that in some cases models maintained or even improved their performance when shuffling marker positions prior to training suggests that CNNs' inductive biases may not be appropriate for some traits. However, CNNs did perform well on other traits, most notably in Holstein cattle Milk Yield prediction. This suggests that detecting patterns in genomic sequence regardless of their position may be benefitial in some contexts. Evaluating which trait architectures' characteristics may favour convolutional models is subject of further research.

Alternatively, we explored constructing 2d feature maps from input signals using dimensionality reduction techniques, and then training CNNs on these 2-dimensional feature maps. This could potentially enable models to expoit non-local correlations and structure in the genome. However, in the Yeast dataset, using random mappings to construct those feature maps from input signals performed comparably to the most commonly used dimensionality reduction techniques. This was interpreted once again as the inadequacy of these models to discover and exploit signal structure, and in the case of the yeast dataset the absence of non local correlations. Although the use of Fermat distance increased predictive accuracy, presumably by diminishing the effects of the curse of dimensionality, the models obtained did not outperform the baselines.

In order to try to exploit population structure, we also formulated complex trait prediction as a node regression problem on a population graph. An association network between individuals was constructed using correlations between individuals. Graph neural networks trained on these graphs achieved state of the art results in Holstein Milk yield prediction. However, as observed in saliency maps and transfer experiments, to what extent the model exploit neighbourhood information and population structure, and how graph topology affects predictions still remains unclear. Training a GNN and CNN joint model achieved the best predictive correlation for this trait. To the best of our knowledge, this is the first Geometric Deep Learning approach to complex trait prediction. How these results generalize to other datasets and traits demands further research.

Pearson correlation is commonly used to evaluate model performance. We

found that optimising it directly, instead of minimising mean squared errors, leads to better model performance. Although this loss does not penalise learning an affine transformation of actual phenotypes, we showed that this affine transformation could be estimated from train data, and led to models with both lower MSE and higher predictive correlations. As far as we are concerned this approach was yet unexplored, and therefore studying how this extends to other models and datasets requires further work. In addition, in the light of these results, proposing and evaluating alternative loss functions in the context of complex trait prediction is a promising research direction.

This page intentionally left blank.

Appendix A

Pipeline

According to recent polls (Figure8, 2018; Kaggle, 2019) data scientists and machine learning practitioners do *not* spend most of their time mining data or adjusting and refining algorithms. Instead, most declare that acquiring, preprocessing, and managing data are the most time consuming tasks. As a result, building a solid data processing pipeline is a key step in any data science project. This is a particularly complex project in that sense, owing to several factors.

To begin with, various datasets were employed. Although standarised formats for genomic data exist (Purcell et al., 2007), none has gained widespread adoption in the agronomic prediction community. Thus, automating data format conversion and preprocessing was particularly important in this context.

Since benchmarking a number of models for each dataset and doing extensive hyperparameter searches is highly demanding on computing power terms, running locally on desktop machines was not feasible. Programming for highly parallelized, distributed and concurrent hardware platforms can be challenging.

Integrating a variety of existing models and tools also makes software dependency management complex, especially in hosted runtimes. Lastly, this thesis is part of an ongoing project that involves many colaborators. Therefore, logging and organising experiments needs to be done in a systematic manner. This is also important for reproducibility.

The following section describes the most important features of the pipeline implemented to deal with these issues (see diagram A.1).

A.1 Containers

Linux kernel containers (LXC) and similar frameworks have become a popular and lightweight alternative to virtual machines for virtualisation (Bernstein, 2014).

Dependency management and cross-platform portability is solved by packaging all dependencies along with the aplication. In contrast to VMs and other hypervisor-based solutions, where virtualization occurs at the hardware level, in containers virtualisation takes place at the operating system level (Merkel, 2014).

Containers have gained popularity among the scientific community seeking to

foster computational reproducibility (Boettiger, 2015).

Several *Docker* images developed for this project can be found on Dockerhub[1]. They were created to meet different dependencies, hardware resources and use cases. For instance, there are images with and without GPU support.

A.2 ClusterUy

With the recent increase of genomic data availability and volume, exploiting high performance computing tools (Zheng et al., 2012), dedicated hardware accelerators and parallel architectures (Cebamanos, Gray, Stewart, & Tenesa, 2014), is becoming increasingly necessary in the field.

A collaborative scientific HPC infrastructure in Uruguay called Cluster-UY (Nesmachnow & Iturriaga, 2019) was used. SLURM (Yoo, Jette, & Grondona, 2003) is used as a workload manager, and singularity (Kurtzer, Sochat, & Bauer, 2017) containers, which are docker compatible, are supported.

Bash scripts for automating runs and batching jobs for most common tasks were implemented, including data preparation, training and testing models.

A.3 Parameter configuration and logging

There are various user defined parameters all along the data processing pipeline ranging from data format, encoding and imputation, to grids for hyperparameter searches. In order to save, load and easily modify settings for a given experiment JSON files were used.

A.4 Logging and visualising experiments online

Unlike traditional software, machine learning systems are built around experimentation (Zaharia et al., 2018). They are highly dependent on multiple inputs such as dataset versions, model hyperparameters and preprocessing code. Consequently, significant efforts have been devoted to adress experiment tracking and reproducibility issues, and several tools exist (Biewald, 2020; Dunn, 2016; Hermann & Del Balso, 2017; Zaharia et al., 2018).

Comet-ml [2] has been chosen for this project. It is an online experiment logging platform which features multiple data input formats, system metrics and outputs, live user defined Plotly[3] charts, hypeparameter and architecture search tools, along with integrated storage solutions and hosting. Comet provides a Python API for interacting with the server, and a higher level abstraction was implemented in order to facilitate incorporating logging to experiments.

[1] https://hub.docker.com/repository/docker/ihounie/dnai/
[2] https://www.comet.ml/
[3] https://plotly.com/

Visualizing model outputs can help to identify systematic errors, population structure, model biases, and even software bugs. Besides from logging predictions and errors we also plot errors using PCA to represent individuals in a 2-dimensional space, and error histograms to contrast model error distributions and assumptions, as well as assessing predictions.

We also implemented comet custom pannels using Plotly javascript API in order to plot aggregate results live.

A.5 Machine learning libraries

Sklearn (Pedregosa et al., 2011) provides implementations of many standard machine learning algorithms, preprocessing, hyperparameter search and evaluation tools. A GPU accelerated implementation of Support Vector Regression (Wen, Shi, Li, He, & Chen, 2018) was also used. Optuna[4], in particular its Bayesian hyperparameter tuning framework was also used.

A.6 Deep learning libraries

The availability of specialised hardware and open source libraries in this field have fueled its rapid growth (while probably hampering other research directions (Hooker, 2020)). The two most popular deep learning libraries were used: Tensorflow[5] and Pytorch[6]. As for the latter, we used the lightning[7] wrapper for all models and the PyTorch Geometric[8] library for Graph Neural Networks.

A.7 Integrating R

Domain specific software libraries have been developed by genetic prediction researchers including BGLR (Pérez & de Los Campos, 2014), rrBLUP (Endelman, 2011), synbreed (Vazquez, Bates, Rosa, Gianola, & Weigel, 2010), somer (Covarrubias-Pazaran, 2016), QGG (Rohde, Fourie Sørensen, & Sørensen, 2020) and nadiv (Wolak, 2012), among others.

Most of them are implemented on R (R Core Team, 2020). In order to leverage this implementations while keeping the python-sklearn pipeline, scikit-learn estimators were implemented following the sklearn API and design (Buitinck et al., 2013). R2py[9] was used to run R embedded in Python processes.

This allowed to incorporate bayesian models and use scikit-learn hyperparameter search and evaluation tools, along with other models.

[4]https://optuna.org/
[5]https://www.tensorflow.org/
[6]https://pytorch.org/
[7]https://www.pytorchlightning.ai/
[8]https://github.com/rusty1s/pytorch$_g eometric$
[9]https://rpy2.github.io/

Appendix A. Pipeline

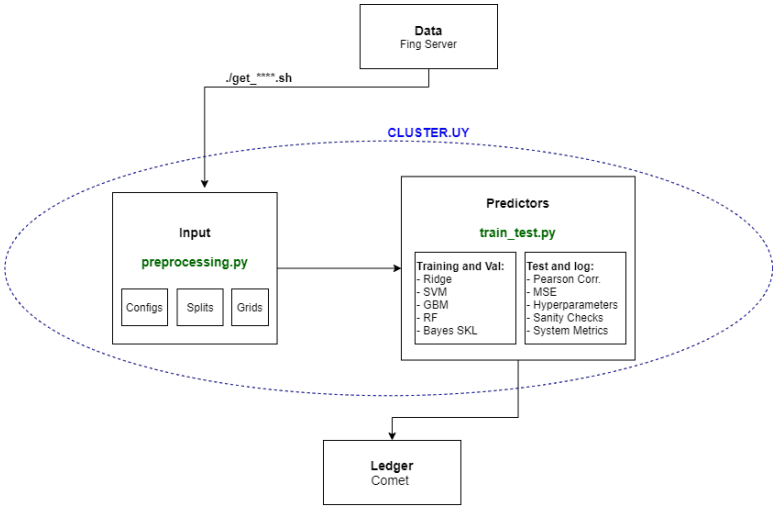


Figure A.1: Diagram of the machine learning pipeline implemented.

Appendix B

Marker clustering

B.1 Introduction

Before addressing the supervised task at hand, we propose an exploratory analysis of the datasets. The main objective is to gain a better understanding of their structure and characteristics. To do so, we will use a hierarchical clustering algorithm to cluster markers according to their similarity. We use an agglomerative approach (i.e. bottom-up). Each observation begins in its own cluster; in each iteration, the two most similar clusters are merged to form a new cluster. This iterative process results in a hierarchy of clusters, in which the cluster at the top (i.e. the root of the tree) contains all observations.

B.2 Metrics and methodology.

We use Pearson's correlation as the distance metric between markers, also called linkage criteria. More specifically, given an input genotype matrix X of N individuals with p markers each, we treat each marker as a variable with N observations. Thus, the correlation coefficient is calculated over these N-dimensional observation vectors.

Several methods exist to compute the similarity between clusters. In our case, we chose the UPGMA (or average) method. Given two clusters u and v, their similarity is given by the following expression:

$$d(u,v) = \sum_{i=0}^{|u|-1} \sum_{j=0}^{|v|-1} \frac{d(u[i], v[j])}{|u| \times |v|} \tag{B.1}$$

where $|.|$ represents the cardinality operator and d is the distance metric.

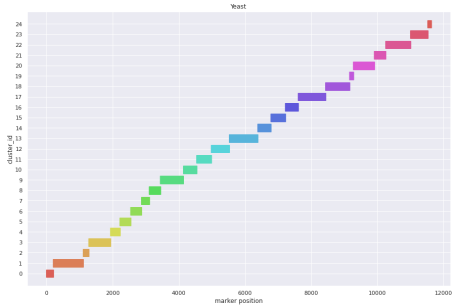
Once the cluster hierarchy is formed, *flat* clusters can be extracted from it. As with linkage criteria, several criteria to extract these clusters exist. We chose SciPy's `maxclust` criterion, which consists of finding a threshold t so that the cophenetic distance between two observations in the same flat cluster is no more

than t while keeping a set number of clusters. The cophenetic distance between two observations is the height of the dendogram where the two branches that contain the two observations merge into a single branch. We experimented with different numbers of maximum clusters for each dataset to account for their varying amount of markers, with larger-dimensional datasets having more flat clusters.

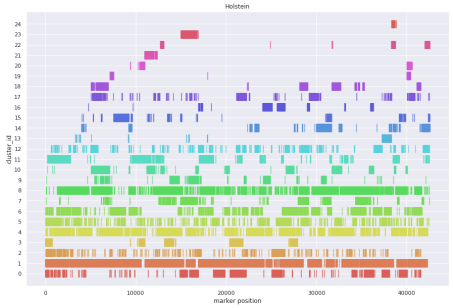
B.3 Results

The graphs in Figure B.1 show the results of the clustering algorithms. For each dataset, we plot each marker's (in the order in which they appear in the genotype matrix) assigned cluster. As evidenced by the plots, the datasets have varying degrees of linkage disequilibrium. On the y axis, the clusters are sorted so that more local clusters are on top (i.e. have a higher cluster id).

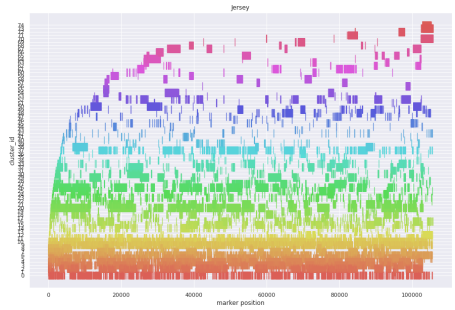
The yeast dataset is the most locally clustered, with all clusters being confined to their own local neighborhood. The Holstein and Jersey markers exhibit weaker spatial correlation, but local clusters can still be distinguished. One the other hand, the wheat dataset does not show significant local clusters. We theorize wheat's lack of spatially correlated markers is due to its already low amount of markers. This could mean that wheat's sampling is more sparse and thus the distance between markers is larger than the distance in the other datasets.



(a) Yeast



(b) Holstein



(c) Jersey

(d) Wheat

Figure B.1: Clustering results.

Appendix C

Machine Learning Models

C.1 Support Vector Regression

Support vector machines (SVMs) were originally proposed by Vapnik and Chervonenkis in 1963 (Chervonenkis, 2013). Back then, SVMs consisted of linear classifiers that maximized the width of the gap between linearly separable classes. Although there exist infinite hyperplanes that separate two linearly separable classes, the *maximum-margin hyperplane* is unique. Intuitively, the maximum-margin hyperplane is the most robust towards noise and thus it achieves better generalization. Consequently, SVMs can be thought of as linear classifiers with automatic regularization.

The problem formulation is different in the context of regression. In this case, a hyperplane is optimal if its predictions deviate from the ground truth by a value no greater than a user-specified constant. Mathematically, given a set of input observations $(\mathbf{x_0}, ..., \mathbf{x_{N-1}}), \mathbf{x_n} \in \mathbb{R}^P$ and target vectors $(y_0, ..., y_{N-1}), y_n \in \mathbb{R}$, the support vector regressor (SVR) fits a hyperplane

$$h(\mathbf{x}) = \mathbf{w^T x} + b$$

where $\mathbf{w} \in \mathbb{R}^{\mathbb{P}}$ and $b \in \mathbb{R}$, subject to the constraint

$$|y_n - \mathbf{w^T x_n} + b| \leq \epsilon, \forall n \tag{C.1}$$

In addition, a regularization penalty is imposed to the weights

$$\mathbf{w^T w} \leq C \tag{C.2}$$

Constraint C.1 implies that an optimal hyperplane may not exist. To solve this issue, the slack variables ξ_n^+ and ξ_n^- are introduced, such that

$$y_n - (\mathbf{w^T x_n} + b) \leq \epsilon + \xi_n^+$$
$$(\mathbf{w^T x_n} + b) - y_n \leq \epsilon + \xi_n^-$$

The purpose of these variables is to loosen up the the constraint imposed by C.1 by allowing a violation of the ϵ margin. The only condition on the slack

variables is $\xi_n^+ \geq 0, \xi_n^- \geq 0$. However, we want the margin's violation to be as small as possible. Thus, the optimization problem can be written as minimizing

$$F(\xi, \xi^*) = \sum_{n=0}^{N-1} \xi_n^+ + \sum_{n=0}^{N-1} \xi_n^- \tag{C.3}$$

subject to

$$\begin{aligned} & y_n - (\mathbf{w^T x_n} + b) \leq \epsilon + \xi_n^+, && \forall n \\ & (\mathbf{w^T x_n} + b) - y_n \leq \epsilon + \xi_n^-, && \forall n \\ & \xi_n^+ \geq 0, && \forall n \\ & \xi_n^- \geq 0, && \forall n \end{aligned} \tag{C.4}$$

and the regularization constraint C.2. This convex optimization problem can be reduced to a quadratic optimization problem by instead minimizing

$$\Phi(\mathbf{w}, \xi, \xi^*) = \frac{1}{2}\mathbf{w^T w} + C' \left(\sum_{n=0}^{N-1} \xi_n^+ + \sum_{n=0}^{N-1} \xi_n^- \right) \tag{C.5}$$

subject to constraints C.4 (Vapnik, 1995). In this case, the parameter C' controls the importance assigned to margin violations (larger C' values impose a larger penalty). The objective function denoted by C.5 together with the inequality constraints in C.4 are known as the primal optimization problem. The Karush-Kühn-Tucker (KKT) Theorem states that a solution $\mathbf{u}^*$ to the primal is optimal if and only if the solution $(\mathbf{u}^*, \boldsymbol{\alpha}^*)$ is a solution to the dual problem

$$\max_{\boldsymbol{\alpha} \geq 0} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})$$

where $\boldsymbol{\alpha}$ denotes the vector of Lagrange multipliers (one for each constraint). For this optimization problem, the Lagrangian is

$$\begin{aligned} \mathcal{L}(\mathbf{w}, b, \xi, \xi^{'}, \alpha, \beta, \gamma^+, \gamma^-) = & \frac{1}{2}\mathbf{w^T w} + C' \left(\sum_{n=0}^{N-1} \xi_n^+ + \xi_n^- \right) \\ & + \sum_{n=0}^{N-1} \alpha_n [y_n - (\mathbf{w^T x_n} + b) - \epsilon - \xi_n^+] \\ & + \sum_{n=0}^{N-1} \beta_n [(\mathbf{w^T x_n} + b) - y_n - \epsilon - \xi_n^-] \\ & + \sum_{n=0}^{N-1} -\gamma_n^+ \xi_n^+ + \sum_{n=0}^{N-1} -\gamma_n^- \xi_n^- \end{aligned}$$

Moreover, the third condition of the KKT Theorem states that

$$\nabla_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\alpha}) \mid_{\mathbf{u}=\mathbf{u}^*, \boldsymbol{\alpha}=\boldsymbol{\alpha}^*} = 0$$

which implies

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} + \sum_{n=0}^{N-1}(\beta_n - \alpha_n)\mathbf{x_n} = 0 \Longrightarrow \mathbf{w} = \sum_{n=0}^{N-1}(\alpha_n - \beta_n)\mathbf{x_n} \tag{C.6}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{n=0}^{N-1}\beta_n - \alpha_n = 0 \Longrightarrow \sum_{n=0}^{N-1}\beta_n = \sum_{n=0}^{N-1}\alpha_n \tag{C.7}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_n^+} = C' - \alpha_n - \gamma_n^+ = 0 \Longrightarrow \gamma_n^+ = C' - \alpha_n \tag{C.8}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_n^-} = C' - \beta_n - \gamma_n^- = 0 \Longrightarrow \gamma_n^- = C' - \beta_n \tag{C.9}$$

Replacing $\mathbf{w}$, γ_n^+ and γ_n^- in the Lagrangian simplifies the dual problem to

$$\begin{aligned}\mathcal{L}(\alpha, \beta) = &-\epsilon \sum_{n=0}^{N-1}(\beta_n + \alpha_n) + \sum_{n=0}^{N-1} y_n(\alpha_n - \beta_n) \\ &- \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}(\alpha_n - \beta_n)(\alpha_m - \beta_m)\mathbf{x_n^T x_m}\end{aligned} \tag{C.10}$$

Since maximizing $\mathcal{L}(\alpha, \beta)$ is equivalent to minimizing $-\mathcal{L}(\alpha, \beta)$, the optimization problem can be re-written as

$$\begin{aligned}\min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \quad & -\mathcal{L}(\alpha, \beta) \\ \text{subject to} \quad & \sum_{n=0}^{N-1}\beta_n = \sum_{n=0}^{N-1}\alpha_n \\ & 0 \leq \alpha_n \leq C' \\ & 0 \leq \beta_n \leq C'\end{aligned}$$

where the two last conditions arise from $\gamma_n^+ \geq 0$, $\gamma_n^- \geq 0$. This is yet another quadratic convex optimization problem, but it is computationally simpler. Quadratic convex problems can be easily solved via quadratic programming, for which several libraries exist.

In addition, the second condition of the KKT theorem (complementary slackness) states the optimal solution must satisfy

$$\alpha_n^*(\epsilon + \xi_n^{+*} - y_n + \mathbf{w}^{*\mathbf{T}}\mathbf{x_n} + b^*) = 0 \tag{C.11}$$

$$\beta_n^*(\epsilon + \xi_n^{-*} - y_n + \mathbf{w}^{*\mathbf{T}}\mathbf{x_n} + b^*) = 0 \tag{C.12}$$

$$\gamma_n^{+*}\xi_n^{+*} = 0 \tag{C.13}$$

$$\gamma_n^{-*}\xi_n^{-*} = 0 \tag{C.14}$$

If $a_n^* > 0$, then from equation C.11 $\epsilon + \xi_n^{+*} - y_n + \mathbf{w}^{*\mathbf{T}}\mathbf{x_n} + b^* = 0$ and hence

$$\mathbf{w}^{*\mathbf{T}}\mathbf{x_n} + b^* - y_n = \epsilon + \xi_n^{+*} \tag{C.15}$$

If $\gamma_n^{+*} > 0$, then from equations C.8 and C.13

$$\alpha_n^* < C' \tag{C.16}$$

$$\xi_n^{+*} = 0 \tag{C.17}$$

which combined with equation C.15 implies that if $0 < \alpha_n^* < C'$

$$\mathbf{w}^{*\mathbf{T}}\mathbf{x_n} + b^* - y_n = \epsilon \tag{C.18}$$

In other words, vectors $\mathbf{x_n}$ whose Lagrange multipliers satisfy $0 < \alpha_n^* < C'$ lie on the boundary of the ϵ-margin. Since they "support" the ϵ-margin, they are called *margin support vectors*. On the other hand, if $\alpha_n^* = C'$ then $\gamma_n^{+*} = 0$ which implies that $\xi_n^{+*} > 0$. These vectors violate the ϵ-margin and are called *non-margin support vectors*. Lastly, vectors with $\alpha_n^* = 0$ lie within the ϵ-margin.

The analysis above is identical for β_n^* and γ_n^{-*}. Consequently, follows from equations C.6 and C.18 it follows that the optimal hyperplane is completely determined by the margin and non-margin support vectors. Hence the name of this algorithm, the *support vector regressor*.

The SVR is a powerful algorithm, but it can only produce linear hypotheses (hyperplanes). To address this issue, the inputs can be transformed into a higher dimensional space via a non-linear transform $\mathbf{\Phi}(\mathbf{x_n}) : \mathbb{R}^P \longrightarrow \mathbb{R}^D$. The SVR can fit the optimal hyperplane in the higher dimensional space, producing a non-linear hypothesis in the original space. Naturally, this dimensional lifting increases the expressive power of the model considerably.

Throughout the SVR fitting procedure, the only step that depends on the input's dimension is calculating the inner product $\mathbf{x_n^T x_m}$ in the Lagrangian (equation C.10). After the transform, this product becomes $\mathbf{\Phi}(\mathbf{x_n})^\mathbf{T}\mathbf{\Phi}(\mathbf{x_m})$, which might become computationally expensive for a high enough D. Fortunately, this problem can be bypassed by the *kernel trick*.

In this context, a kernel is a continuous symmetric positive semidefinite function $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$. The mathematical basis for the kernel trick is Mercer's Theorem, which states that all continuous symmetric functions can be expressed as an inner product

$$K(\mathbf{x_n}, \mathbf{x_m}) = \mathbf{\Phi}(\mathbf{x_n})^\mathbf{T}\mathbf{\Phi}(\mathbf{x_m})$$

for some $\mathbf{\Phi}$ if and only if K is positive semidefinite. Thus, if $\mathbf{\Phi}(\mathbf{x_n})^\mathbf{T}\mathbf{\Phi}(\mathbf{x_m})$ is a valid inner product in $\mathcal{V}$, there exists a kernel function K which can calculate this inner product in another vector space $\mathcal{X}$.

In this case, the $\mathcal{V}$ space corresponds to the higher dimensional feature space $\mathbb{R}^D$. The gist of the kernel trick is that for certain $\mathbf{\Phi}$ transforms, the kernel function can be efficiently computed in the original $\mathbb{R}^P$ space. For example, consider the second order polynomial transform

$$\mathbf{\Phi}(\mathbf{x}) = (1, x_0, x_1, ..., x_{p-1}, x_0x_0, x_1x_1, ..., x_{p-1}x_{p-1})$$

where $D = 1 + P + P^2$. The computation of $\mathbf{\Phi}(\mathbf{x})$ takes $O(D)$ time, and thus

$$\mathbf{\Phi}(\mathbf{x_n})^\mathbf{T}\mathbf{\Phi}(\mathbf{x_m}) = 1 + \sum_{i=0}^{p-1} x_{ni}x_{mi} + \sum_{i=0}^{p-1}\sum_{j=0}^{p-1} x_{ni}x_{nj}x_{mi}x_{mj}$$

also takes $O(D)$ time to compute. However, reorganizing the double summation we have that

$$\sum_{i=0}^{p-1}\sum_{j=0}^{p-1} x_{ni}x_{nj}x_{mi}x_{mj} = \left(\sum_{i=0}^{p-1} x_{ni}x_{mi}\right) \times \left(\sum_{j=0}^{p-1} x_{nj}x_{mj}\right) = (\mathbf{x_n^T x_m})^2$$

Therefore, the inner product in $\mathbb{R}^D$ can be calculated as

$$K_{\text{poly}}(\mathbf{x_n}, \mathbf{x_m}) = 1 + \mathbf{x_n^T x_m} + (\mathbf{x_n^T x_m})^2$$

which takes $O(P)$ to compute. The derivation of the second order polynomial kernel can be extended to the general Q-degree polynomial kernel

$$K_{\text{poly}}(\mathbf{x_n}, \mathbf{x_m}) = (\zeta + \gamma \mathbf{x_n^T x_m})^Q$$

where $\zeta > 0, \gamma > 0$ and $Q \in \mathbb{N}$. Thus, the inner product in equation C.10 can be efficiently computed in the input feature space even after the high dimensional, non-linear polynomial transform. This makes the kernel trick an extremely powerful tool which enables the SVR to learn complex hypotheses, while preserving its automatic regularization properties.

Another popular kernel is the Gaussian radial basis function (RBF), which has the form

$$K_{\text{RBF}}(\mathbf{x_n}, \mathbf{x_m}) = \exp\left(-\theta \|\mathbf{x}_i - \mathbf{x}_j\|_2^2\right)$$

with $\theta > 0$. What makes this kernel special is that it allows a projection into an infinite-dimensional feature space. This property is derived from the following analysis:

$$\begin{aligned}
K_{\text{RBF}}(\mathbf{x_n}, \mathbf{x_m}) &= \exp\left(-\gamma \|\mathbf{x_n} - \mathbf{x_m}\|^2\right) \\
&= \exp\left(-\gamma (\mathbf{x_n} - \mathbf{x_m})^{\mathbf{T}} (\mathbf{x_n} - \mathbf{x_m})\right) \\
&= \exp\left(-\gamma \left[\mathbf{x_n^T}(\mathbf{x_n} - \mathbf{x_m}) - \mathbf{x_m^T}(\mathbf{x_n} - \mathbf{x_m})\right]\right) \\
&= \exp\left(-\gamma \left[\mathbf{x_n^T x_n} - \mathbf{x_n^T x_m} - \mathbf{x_m^T x_n} + \mathbf{x_m^T x_m}\right]\right) \\
&= \exp\left(-\gamma \left[\mathbf{x_n^T x_n} + \mathbf{x_m^T x_m} - 2\mathbf{x_n^T x_m}\right]\right) \\
&= \exp\left(-\gamma \|\mathbf{x_n}\|^2\right) \exp\left(-\gamma \|\mathbf{x_m}\|^2\right) \exp(2\gamma \mathbf{x_n^T x_m})
\end{aligned}$$

For convenience, define the A constant such that

$$A = \exp\left(-\gamma \|\mathbf{x_n}\|^2\right) \exp\left(-\gamma \|\mathbf{x_m}\|^2\right)$$

then

$$\begin{aligned}
K_{\text{RBF}}(\mathbf{x_n}, \mathbf{x_m}) &= A \exp(2\gamma \mathbf{x_n^T x_m}) \\
&\overset{*}{=} A \sum_{r=0}^{\infty} \frac{\left(2\gamma \mathbf{x_n^T x_m}\right)^r}{r!} \\
&= A \sum_{r=0}^{\infty} \frac{(2\gamma)^r \left(\mathbf{x_n^T x_m}\right)^r}{r!} \\
&= A \sum_{r=0}^{\infty} \frac{(2\gamma)^r}{r!} \times K_{\text{poly(r)}}(\mathbf{x_n^T x_m})
\end{aligned}$$

where $*$ comes from the Taylor expansion of the exponential function. This result indicates the RBF kernel can be viewed as an infinite sum of polynomial kernels with increasing dimensions. The $r!$ denominator implies that higher-dimensional kernels contribute less to the solution than lower-dimensional ones. However, the attenuation of higher-dimensional kernels can be controlled with the γ parameter, with higher γ leading to more complex hypotheses.

C.2 Decision Tree Ensembles

.

Model ensembles consist of combining different models (weak learners) in order to obtain a better performance than any of the individual models'. Thus, ensembling can be thought of as a way of compensating for sub-optimal models with extra computation. Although the ensemble model represents a single function, it does not necessarily belong to the same function class as the models from which its built. For example, a simple ensemble model could be a weighted average between the predictions of a regularized linear regressor and a support vector regressor.

In tree-based ensembling, the weak learners consist of decision trees (DTs). DTs are a non-parametric supervised learning algorithm which predicts the target variable by learning simple decision rules from the training data.

In the context of regression, given a set of input features $(\mathbf{x_0}, ..., \mathbf{x_{N-1}}), \mathbf{x_n} \in \mathbb{R}^P$ and target vectors $(\mathbf{y_0}, ..., \mathbf{y_{N-1}}), \mathbf{y_n} \in \mathbb{R}^D$, a DT recursively partitions the feature space such that samples belonging to the same partition have similar targets. Each of these partitions is represented by a node Q_m in the tree, containing a subset of N_m training samples. In the case of binary trees, a split $\theta = (x_p, t)$ is defined by a feature x_p and a threshold t (for numerical features) or a subset of categories A (for categorical features). Thus, a node Q_m is partitioned into nodes $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ by split θ such that

$$\begin{cases} Q_m^{left}(\theta) = \{(\mathbf{x_m}, \mathbf{y_m}) : x_{mp} \leq t, (\mathbf{x_m}, \mathbf{y_m}) \in Q_m\} \\ Q_m^{right}(\theta) = Q_m \backslash Q_m^{left}(\theta) \end{cases}$$

for numerical features and

$$\begin{cases} Q_m^{left}(\theta) = \{(\mathbf{x_m}, \mathbf{y_m}) : x_{mp} \in A, (\mathbf{x_m}, \mathbf{y_m}) \in Q_m\} \\ Q_m^{right}(\theta) = Q_m \backslash Q_m^{left}(\theta) \end{cases}$$

for categorical features. The algorithm follows a greedy behavior, which means each node is split according to its best possible split. The quality of a split is measured using a loss function $H(Q)$

$$G(Q_m, \theta) = \frac{N_m^{left}}{N_m} H(Q_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(Q_m^{right}(\theta))$$

where G represents the splits' impurity. Thus, the best possible split at node Q_m is

$$\theta^* = \text{argmin}_\theta \, G(Q_m, \theta)$$

The process is then repeated for the new nodes Q_m^{left} and Q_m^{right} until the stopping criteria are met. The algorithm begins from the root node, which contains all of the training samples. Without any restriction, the DT could grow until all nodes have constant target values. However, these deep trees suffer from severe overfitting, so additional stopping criteria (defined by the user) are also implemented. These criteria include maximum tree depth, minimum number of samples per node, and minimum impurity decrease between parent and child nodes. In the context of genomic prediction, shallower decision trees perform more favorably on most datasets (Azodi et al., 2019; Goldstein et al., 2010).

For prediction, the tree assigns input vectors to one of the terminal nodes (also called leaf nodes) according to the learned splits θ. The predicted value for an input belonging to node Q_M is the node's mean vector, although the median can also be used

$$\hat{\mathbf{y}_\mathbf{M}} = \frac{1}{N_M} \sum_{\mathbf{y} \in Q_M} \mathbf{y}$$

Several loss functions can be used for regression tasks, such as the mean absolute or squared error. When the mean squared error is optimized, the loss function takes the form of

$$H(Q_m) = \frac{1}{N_m} \sum_{\mathbf{y} \in Q_m} (\mathbf{y} - \overline{\mathbf{y_m}})^2$$

which is equivalent to minimizing the node's variance.

In practice, individual DTs are heavily dependent on the dataset they are trained on. A small change in the training data can result in considerably different splits and thus predictions. Ensembling naturally addresses this issue by combining several DTs into a single model, improving its generalizability and robustness.

Ensemble methods generally fall into one of two categories: averaging methods and boosting methods. Averaging methods build each weak learner independently and then average their predictions. Thus, the ensembled model's variance is significantly lower than any of the weak learner's. On the other hand, boosting methods build the weak learners sequentially instead of independently. Each consecutive learner improves upon the last, which can be thought of as minimizing the ensemble model's bias. To do this, gradient boosting methods begin from a constant function F_0 such that

$$F_0(\mathbf{x}) = \text{argmin}_\gamma \sum_{i=0}^{N-1} L(\mathbf{y_i}, \gamma)$$

where L is the loss function to be optimized (which needs to be differentiable). In the case of regression with mean squared error loss, $F_0 = \overline{\mathbf{y}}$. Each gradient boosting iteration adds a new model to the previous ensemble

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + h_m(\mathbf{x}) = \mathbf{y} \Longrightarrow h_m(\mathbf{x}) = \mathbf{y} - F_{m-1}(\mathbf{x})$$

From the previous expression, it follows that

$$h_m^* = \text{argmin}_{h_m} \sum_{i=0}^{N-1} L(\mathbf{y_i}, F_{m-1} + h_m(\mathbf{x_i}))$$

In practice, finding the best possible function at each iteration is computationally infeasible. However, finding an approximation of this function is good enough, since any errors will be corrected by the next ensemble (and so on). Using the first-order Taylor expansion of $L(\mathbf{y_i}, F_{m-1}(\mathbf{x_i}) + h_m(\mathbf{x_i}))$ at $F(\mathbf{x_i}) = F_{m-1}(\mathbf{x_i})$, we have

$$L(\mathbf{y_i}, F_{m-1}(\mathbf{x_i}) + h_m(\mathbf{x_i})) \approx L(\mathbf{y_i}, F_{m-1}(\mathbf{x_i})) + h_m(\mathbf{x_i}) \left[\frac{\partial L(\mathbf{y_i}, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})} \right]_{F=F_{m-1}}$$

Thus, the optimal h_m is

$$h_m^* = \text{argmin}_{h_m} \sum_{i=0}^{N-1} h_m(\mathbf{x_i}) \left[\frac{\partial L(\mathbf{y_i}, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})} \right]_{F=F_{m-1}}$$

The term $\left[\frac{\partial L(\mathbf{y_i}, F(\mathbf{x_i}))}{\partial F(\mathbf{x_i})} \right]_{F=F_{m-1}}$ is the derivative of the loss w.r.t. the predicted value, evaluated at the previous ensemble model's prediction (also known as pseudo-residuals). Since the loss is differentiable, this term is easy to compute in closed form. Consequently, the optimal solution h_m^* is proportional to the negative gradient of the loss. This implies that GBMs can be thought of as iterative gradient descent algorithms which optimize the loss over the function space.

In our case, we explored two DT based ensemble methods: Gradient Boosting Machines (GBMs, also called TreeBoost models) (Friedman, 2001) and Random Forests (RFs) (Breiman, 2001). As their name indicates, the former belong to the boosting category, while the latter belong to the averaging methods category.

GBMs are gradient boosting algorithms in which the weak learners are DTs. On the other hand, RFs inject randomness into the algorithm by constructing several DTs which fit different subsets of the whole training dataset. Moreover, the best split at each iteration is determined in a random subset of features. These two sources of randomness imply that DTs have reasonably independent errors, which cancel out during the averaging process. Thus, the variance of the RF is greatly reduced, at the cost of a slight increase in bias.

Appendix D

SNP graph representation

Throughout this work we focused mainly on individuals as nodes. Topology inference using SNPs as nodes is more challenging since $n << p$. This appendix introduces some techniques and important considerations in this scenario, as presented by Kolaczyk and Csárdi (2020).

D.1 Partial Correlations

When constructing association networks, partial correlations are often preferred, from the stance that edges should represent direct *associations* or direct effects between vertices. That is, when considering two variables the effect of the remaining variables should be adjusted. It may be noted that in the context of *Gaussian Graphical Models* this is equivalent to *Markovian* or *conditionally independent* Network estimation. Under gaussianity assumptions, partial correlations are proportional to the inverse of the covariance matrix, known as the precision matrix. However, in the case were $n < p$ the covariance matrix is rank deficient and thus not invertible. Several precision matrix estimation methods exist, which overcome this limitation by applying regularization techniques - or equivalently *shrinkage*.

D.2 Covariance Estimation

In the case where $n << p$, applying regularisation may lead to lower MSEs by achieving a better bias-variance tradeoff, a more stable eigenvector estimation, and enforce non-singularity.

Shrinkage estimators consist of a convex combination between the sample covariance and a *shrinkage target*, which is usually a scaled identity matrix - but can be replaced by other structured estimators.

$$\Sigma_{\text{shrunk}} = (1 - \alpha)\hat{\Sigma} + \alpha\mathbf{S}$$

Where $\mathbf{S}$ is the shrinkage target and α is the shrinkage constant.

The identity matrix is usually scaled by the average eigenvalue, i.e. $\mathbf{S} = \frac{\operatorname{Tr}\hat{\Sigma}}{p}\,\mathrm{I}$, thus shrinking eigenvalues towards their mean.

Several methods for finding an appropriate α exist, including Leidot-Wolf and Oracle Approximate Shrinking. The latter assumes data is normally distributed.

D.3 Association inference and Multiple testing

The task of inferring edges representing a *statistically significant* correlation between variables can be posed as a hypothesis test where the null hypothesis is that the variables are uncorrelated:

$$H_0 : \operatorname{corr}(X_i, X_j) = 0 \quad \text{versus} \quad H_1 : \operatorname{corr}(X_i, X_j) \neq 0,$$

Choosing a test statistic and an appropiate null distribution the set of non-zero correlations can be inferred.

Since this must be done for every variable, the problem of multiple testing arises: with high probability true null hypotheses will be rejected. Several techniques exist to limit this effect, including *permutation* methods, *familywise error rate (FWER)* methods (notably the Bonferroni correction), and False Discovery Rate adjustments.

D.4 False Discovery rate adjustment

The False Discovery Rate is defined as the expected rate of type I errors (false rejections):

$$\mathrm{FDR} = \mathbb{E}\left(\frac{R_{\mathrm{false}}}{R} \mid R > 0\right)\mathbb{P}(R > 0)$$

where R is the number of rejections and R_{false} is the number of false rejections.

The Benjamini/Hochberg FDR correction consists of ordering p values and rejecting those which satisfy

$$p_{(i)} \leq (i/m)\gamma,$$

where m is the number of tests, in order to guarantee that (with high probability) $\mathrm{FDR} \leq \gamma$.

Appendix E

Graph Topology Descriptive Analysis

We analyse the topology of the association graphs constructed using Pearson's correlation coefficient and either k-NN or thresholding, as described on Chapter The method used to construct the graph, as well as the choice of parameters, namely k and threshold has a significant impact on the resulting graphs. A descriptive network analysis was then carried out to compare the GSOs produced by the different methodologies mentioned above. The analysis is based on Kolaczyk and Csárdi (2020), where all of the following metrics are described (on Chapter 4). Graph tool (Peixoto, 2014) library was used to compute them, refer to their documentation for further implementation details.

E.1 Connectivity

E.1.1 Components

If an induced subgraph (as defined on Section 7.2) is *connected* it is called a *connected component*. The number of connected components for different networks is shown in Figure E.1. As it can be seen, the *knn* graphs constructed have one connected component, whereas using thresholding small isolated components appear at higher sparsity levels.

Appendix E. Graph Topology Descriptive Analysis

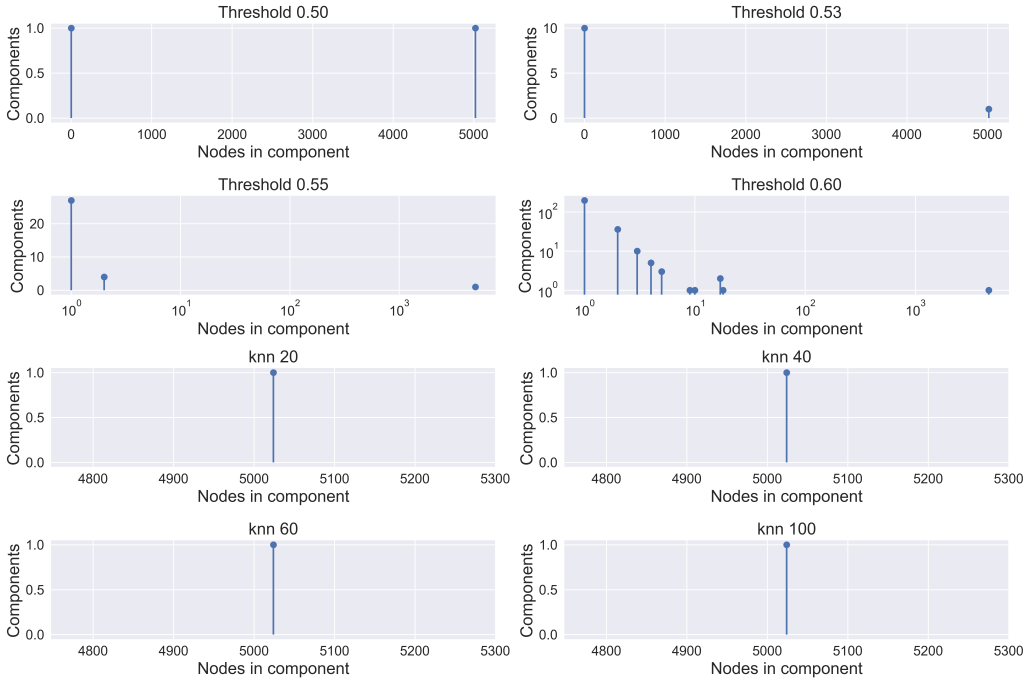


Figure E.1: Component size and number of components in a graph with 5024 nodes, constructed using Pearson correlations and different thresholding levels and number of neighbours.

E.1.2 Isolated Nodes

To further illustrate how connectivity varies with thresholding levels, we include a plot of the number of isolated nodes, i.e. with no neighbours at all, for different thresholding levels. To what extent having isolated nodes is detrimental to the

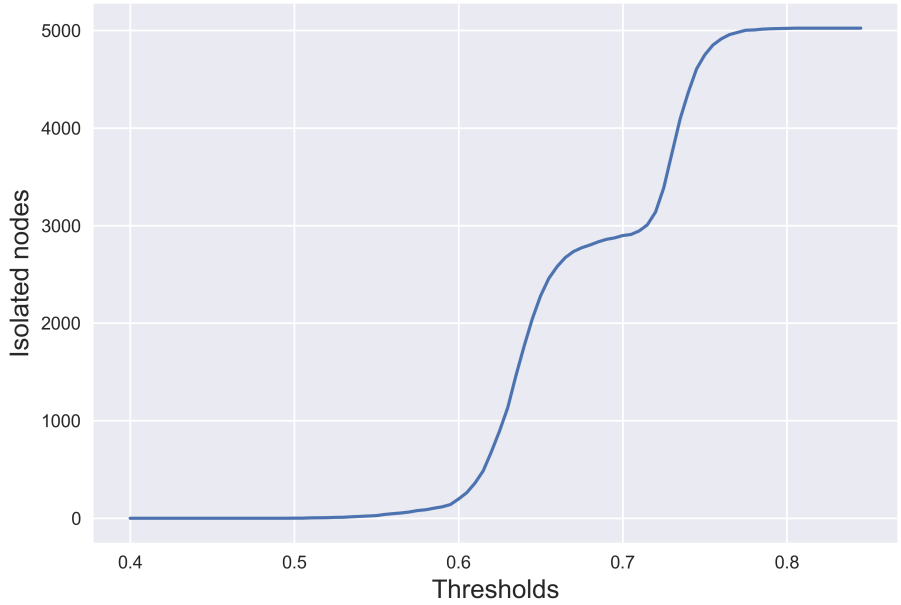


Figure E.2: Number of isolated nodes for a graph with 5024 nodes, constructed using Pearson correlations and different thresholding levels.

task at hand, and whether it reflects the presence of individuals with no close relatives on the dataset, requires further examination.

E.2 Node Centrality

E.2.1 Degree

As defined in the previous section, the degree is the number of incident edges of a node. As E.3 shows, degree histograms present significant variations depending on the thresholding levels. Degree distributions are significantly different from a

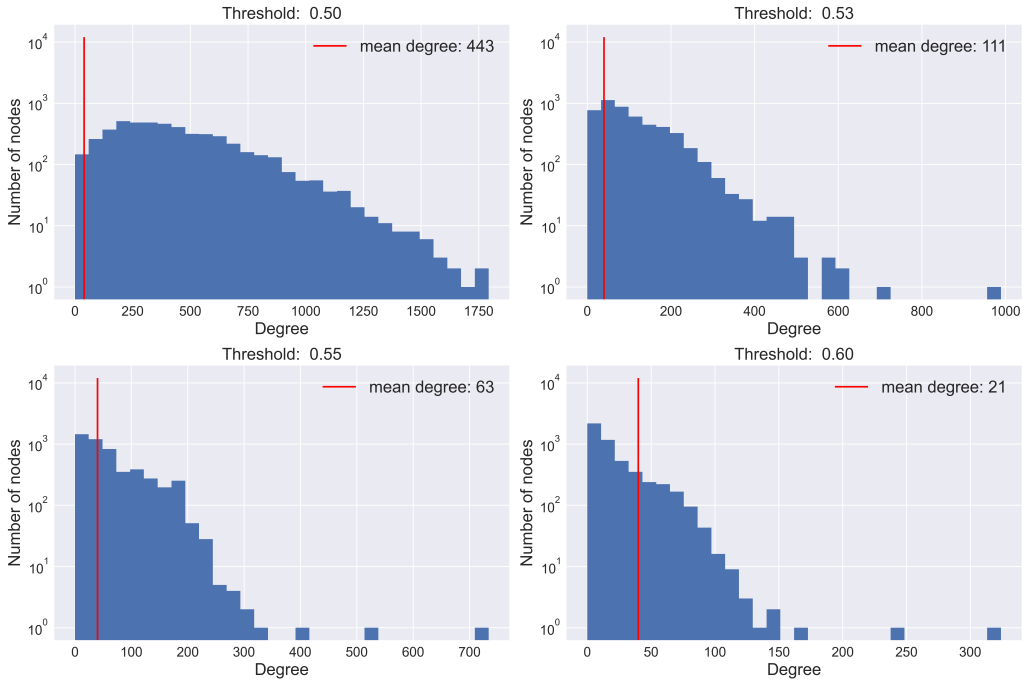


Figure E.3: Node degree histogram and mean degree for a graph with 5024 nodes, constructed using Pearson correlations and different thresholding levels.

knn graph where the same degree is imposed to all nodes. Notably, thresholding produces a large number of nodes with few neighbours and some with large degree.

For instance, using a threshold of 0.55 while there are only five nodes with degree greater than 300, 1378 nodes have degrees under 25. This may be related to population structure.

E.2.2 Betweenness

Betweenness centrality is also related to the cost of taking a node out, by capturing the extent to which a node is in the shortest paths of all other nodes. The most commonly used measure, as defined by Freeman (1977) is:

$$c_B(v) = \sum_{s \neq v \neq t \in \mathcal{V}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of those paths that pass through v. As with other centrality measures, this is an important property of the graphs since not all individuals from the population will be available when training. That is, some nodes will be actually taken out. Therefore, understanding to what extent removing them can affect the spreading of information is especially relevant.

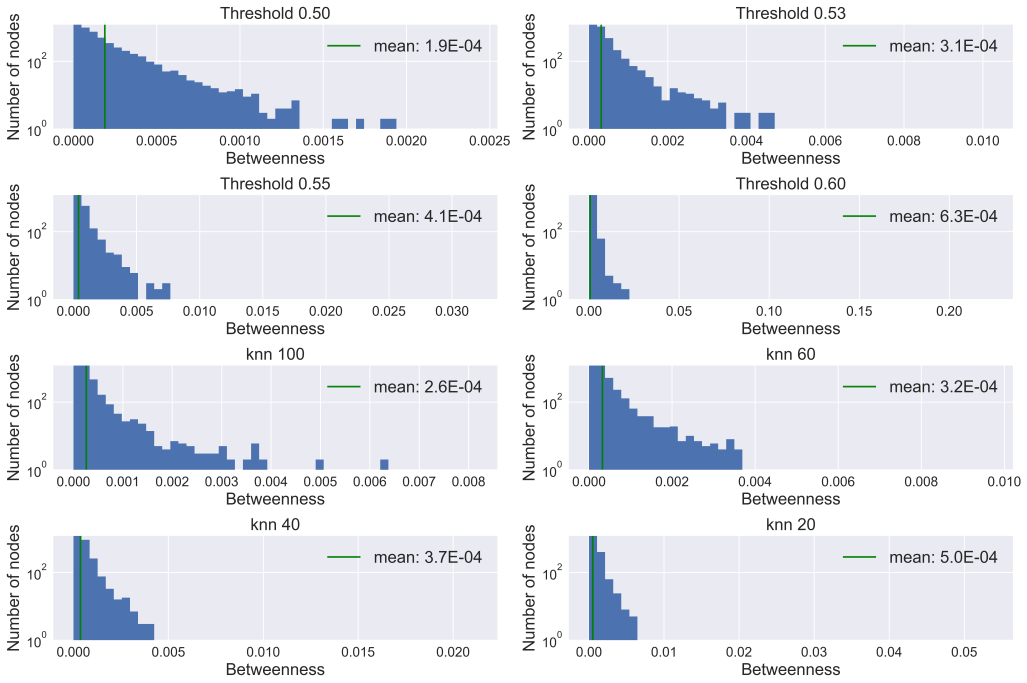


Figure E.4: Betweenness node histogram for a graph with 5024 nodes, constructed using Pearson correlations and different thresholding levels and number of neighbours.

As expected, on sparser graphs there are more nodes with higher betwenness. Nonetheless, the distribution of this metric shows an unexpected degree of similarity between *knn* and *thresholding*. Understanding why this happens and to what extent it affects predictions requires further analysis.

E.2.3 Closenness

Intuitively, it represents the cost of spreading information to all other nodes:

$$c_i = \frac{1}{\sum_j dist(i,j)}$$

where $dist(i,j)$ is the geodesic distance or shortest path (computed using *Dijkstra's algorithm*) from node i to j, and path length is defined as:

$$l(i,j) = \sum_{l=1}^{k} \frac{1}{w_{q_{l-1}q_l}},$$

where $q = i, \ldots, j$ is a sequence of nodes connecting node i and node j. Although in some graphs edge weights may directly represent distances, we take their inverse $\frac{1}{w_{q_{l-1}q_l}}$ since our weights represent similarities. This is related to the Fermat distance between samples introduced in earlier sections (6.4.4). It should be noted that there may be more than one path with minimum length.

In case there is no path between the two nodes, in the graph-tools library implementation closeness is taken to be zero. As we have seen, isolated nodes are more common in thresholded graphs with higher sparsity. Therefore, direct comparisons between *knn* and *thresholding* graphs and different sparsity levels are hindered. As shown in Figure E.5, as a result of the aforementioned phenomenon, nodes with high closeness values appear when thresholding with 0.6. Interestingly, some graphs with similar mean degrees (see 60-nn and 0.53 thresholding) ended up having similar mean closeness. It may be pointed out that when calculating degree we did not take into account weights (just connectivity). For large thresholding levels the impact of nodes with few or no neighbours is clearly visible. Overall, *Knn* graphs showed heavier tails or skewness, which may reflect that nonzero weights in knn graphs present more variation than their thresholded counterparts (although differences in topology could have had the opposite effect on path length). Although this particular metric certainly has its limitations, measures which generalize the notion of closeness to isolated graphs exist (Dangalchev, 2006; Rochat, 2009), which may be more adequate for analyzing sparser graphs.

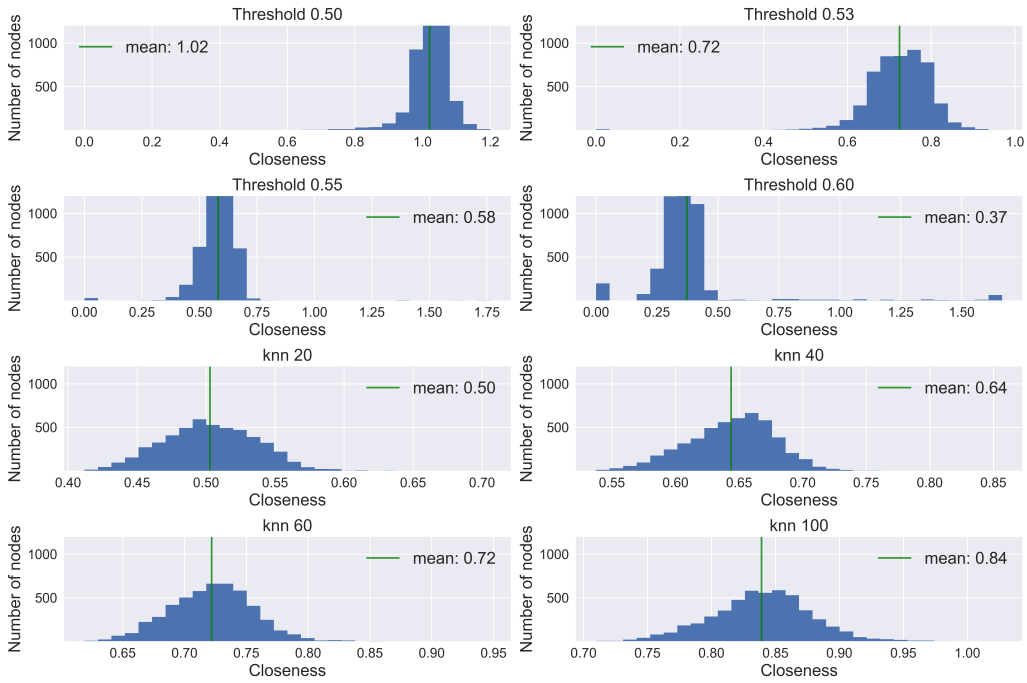


Figure E.5: Closeness node histogram for a graph with 5024 nodes, constructed using Pearson correlations and different thresholding levels and number of neighbours.

E.3 Cohesion

E.3.1 Local Clustering Coefficient

A *complete graph* is a graph where every pair of nodes is joined by an edge, and a *clique* is a *complete subgraph*. The local clustering coefficient c_i, as defined in equation E.1, can be used to measure how far *typical neighbourhoods* are from being a clique (their 'cliqueishness' according to Watts and Strogatz (1998)). It measures the fraction of edges present in the neighbourhood of a node to the number of edges that would be present if said neighbourhood was a clique.

$$c_i = \frac{|\{e_{jk}\}|}{d_i\,(d_i - 1)} : v_j, v_k \in \mathcal{N}(i), e_{jk} \in \mathcal{E}, \tag{E.1}$$

where d_i is the degree of vertex i.

The coefficient is undefined for isolated nodes, and was set to zero in those cases. As in the case of closenss, this may hinder direct comparisons. Figure E.6 shows that for low sparsity levels local clustering coefficients tend to be higher on knn graphs. Conversely, for higher sparsity levels, and without taking into account isolated nodes, clustering coefficients seem to be higher in thresholded graphs.

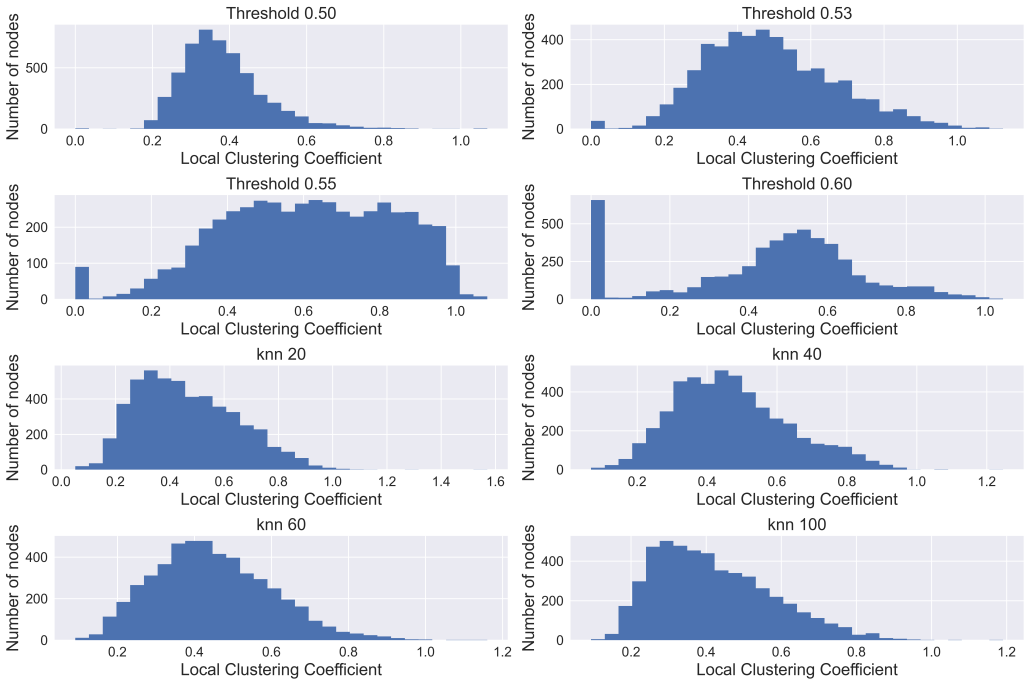


Figure E.6: Local clustering coefficient node histogram, in a graph with 5024 nodes, constructed using Pearson correlations and different thresholding levels and number of neighbours.

E.4 Graph Spectral Analysis

The spectral decomposition of graph shift operators can give useful insights about graph structure. Some of the properties already mentioned could have been analyzed through the Laplacian's spectral decomposition. For instance, the number of connected components is equal to the dimension of the nullspace of the Laplacian (or equivalently the algebraic multiplicity of eigenvalue 0).

The choice of the operator will obviously have an impact on the analysis. Some state that Laplacian eigenvalues are more intuitive and "important" (Mohar, Alavi, Chartrand, & Oellermann, 1991). On the other hand, adjacency spectrum has also been thoroughly studied, often motivated by its connection with random walks on graphs (Chung & Graham, 1997). Since several results linking eigenvalues to interesting graph properties exist for both operators, both could be employed. Regarding differences between normalised and unnormalised operators, in the case of the Laplacian and Adjacency an affine transformation with bounded errors which depend on the minimum and maximum degree of the graph can be found (Lutzeyer & Walden, 2017). We will analyze both the normalized and unnormalised adjacency and laplacian matrices and some common properties in the spectral domain.

E.5 Visualizing Graph Spectrum

We are dealing with Hermitian operators since the graph is undirected. therefore, all eigenvalues are real, which simplifies the analysis.

E.6 Algebraic connectivity

The notion of graph connectivity is usually defined as the minimum number of nodes or edges that would need to be removed to disconnect the graph into two or more components. The Fiedler value is the second smallest eigenvalue of a graph and many results linking it with graph connectivity exist (Brandes, 2005). Perhaps the simplest one is that a graph is connectted iif the fiedler value is not zero. Furthermore, several properties of the graph such as the mean distance, diameter (largest distance) and many others presented on (Brandes, 2005). As shown on Table E.1.

E.7 Spectral Radius Ratio for node degree.

It is a measure node degree variation (Meghanathan, 2014), defined as:

$$sr_{ratio} = \frac{\lambda_p}{d_{avg}},$$

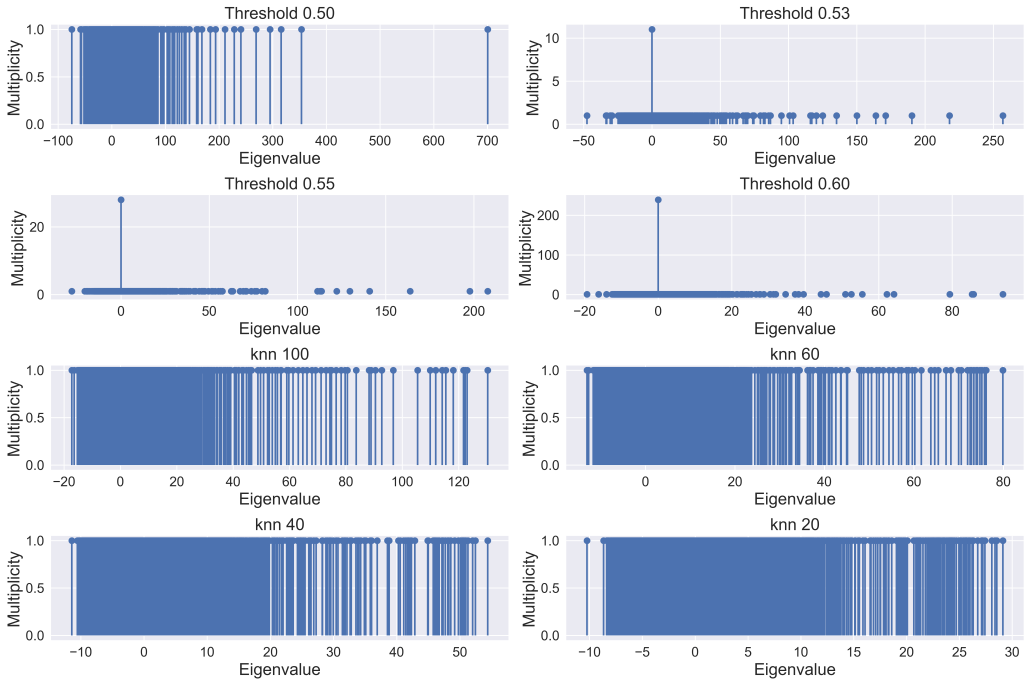


Figure E.7: Eigenvalues and algebraic multiplicity for the adjacency operator.

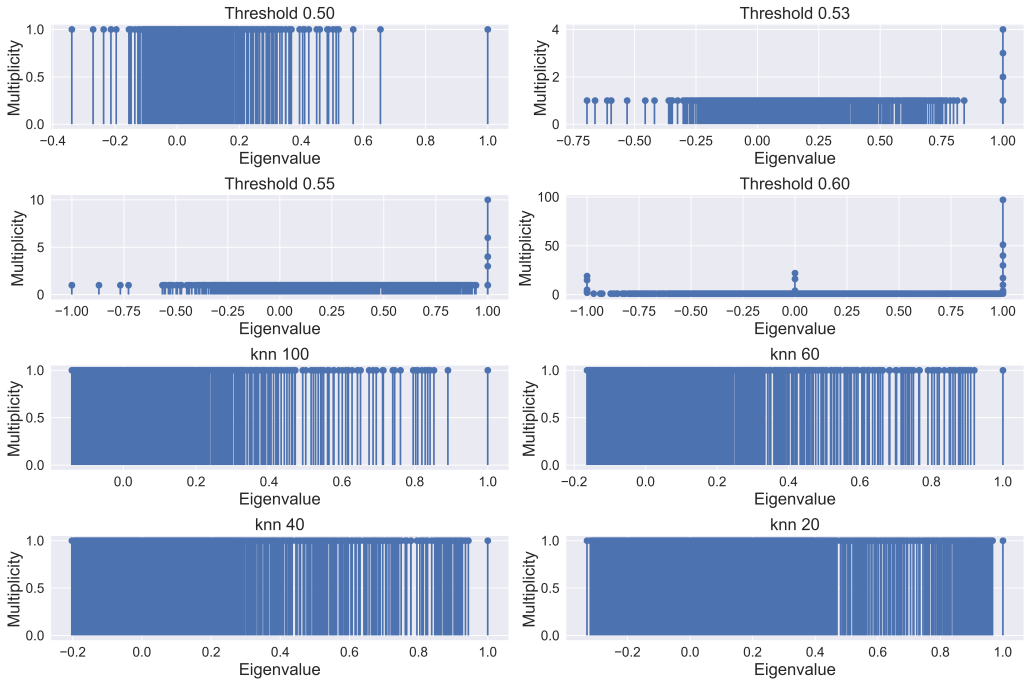


Figure E.8: Eigenvalues and algebraic multiplicity for the normalized adjacency operator.

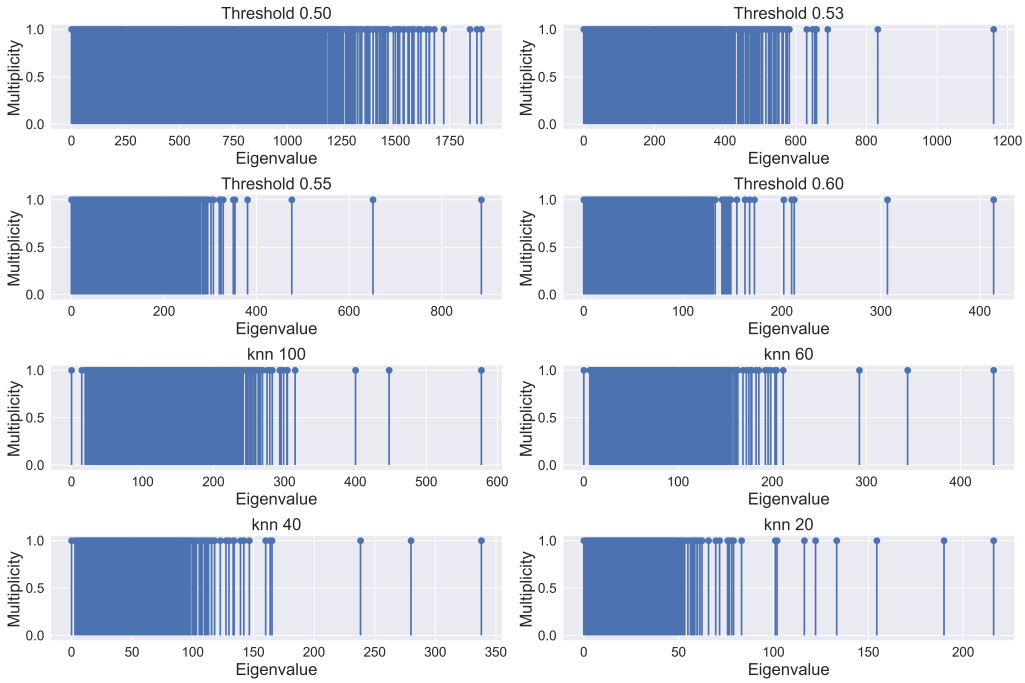


Figure E.9: Eigenvalues and algebraic multiplicity for the Laplacian operator.

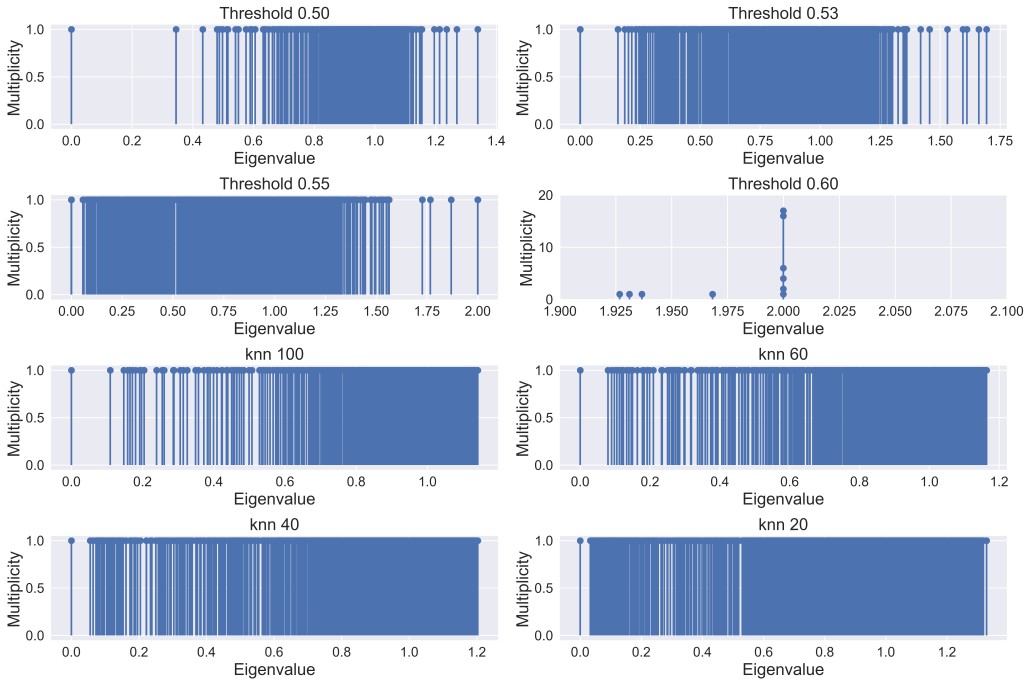


Figure E.10: Eigenvalues and algebraic multiplicity for the normalized Laplacian operator.

		Fiedler	λ_p	d_{avg}	sr_{ratio}
Thresholding	0.50	1.2E-13	1897.5	442.5	4.29
	0.53	1.5E-15	1160.9	111.2	10.44
	0.55	1.0E-15	885.7	63.4	13.97
	0.60	3.9E-31	413.9	21.2	19.51
knn	100	14	577.4	100	5.77
	60	6.3	435.1	60	7.25
	40	2.9	338.0	40	8.45
	20	8.5E-01	216.1	20	10.81

Table E.1: Fiedler eigenvalue, spectral radius, mean degree and their ratio.

where λ_p is the largest eigenvalue, called *principal eigenvalue* or *spectral radius*, and d_{avg} the average degree. It can be shown that:

$$1 \leq d_{min} \leq d_{avg} \leq \lambda_p \leq d_{max},$$

where d_{min} and d_{max} are the minimum and maximum eigenvalues respectively. The farther is the value from 1, the larger the variation in node degree.

As shown on Table E.1, *knn* graphs have a higher algebraic connectivity than *thresholded* graphs.

E.8 Further Analysis.

An exhaustive descriptive analysis would surely include other measures such as local density or assortativity and mixing coefficients. In addition, other techniques such as graph partitioning (in particular hierarchichal clustering), studying edge and node cut or flow properties, or performing density estimation, could help to give a more accurate description of the graphs obtained.

Despite the fact this exploratory analysis can lead to useful insights, the main objective is not to evaluate graph topology inference per se. In that sense, graph topologies should ultimately be evaluated with respect to their performance on the prediction task, together with predictive models supported on those graphs.

Appendix F

Other noise experiments

F.1 Phenotypic Noise.

As when dropping markers (see Section F.2 exhibits high variance between runs. Due to a low number of samples and high dimensionality, model performance may be greatly affected by train-test splits, and by which sample gets contaminated (both chosen randomly).

As it can be seen, the behaviour is similar between Yeast and Holstein dataset. In both, predictive correlation falls in an approximately linear manner as the number of contaminated samples increases. The fact that Wheat exhibits a "noisier" behaviour may owe to the fact that there are significantly fewer samples in that dataset. On the Holstein dataset model performance has less variance. This could be linked to the fact that it has the highest number of samples.

It is worth noting that -mostly- models do not present significant differences in their behaviours.

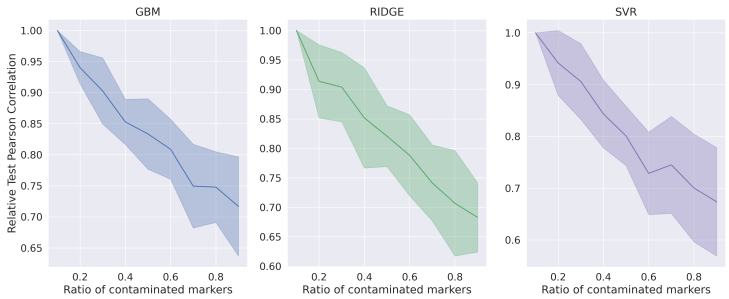


Figure F.1: Performance of different methods in the yeast dataset under varying amounts of contaminated phenotypes.

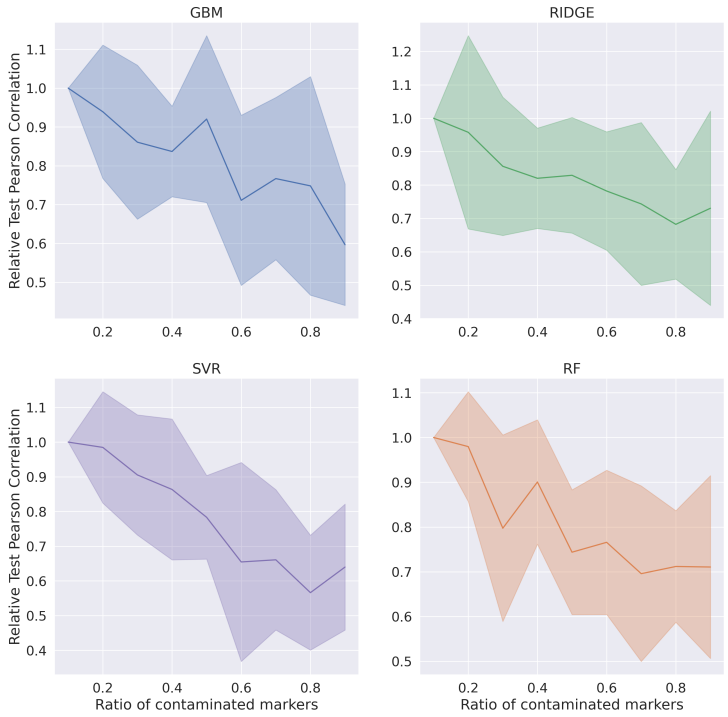


Figure F.2: Performance of different methods in the wheat dataset under varying amounts of contaminated phenotypes.

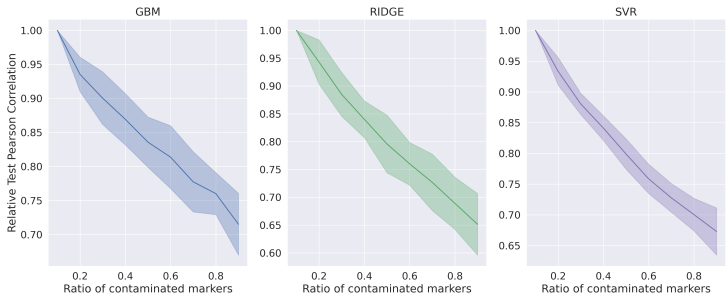


Figure F.3: Performance of different methods in the Holstein dataset under varying amounts of contaminated phenotypes.

F.2 Number of samples.

When dropping samples the Yeast dataset starts with the slower decline in performance, and falls rapidly for extreme values. This may owe to the high *Linkage Disequilibrium* present. That is, since samples are highly correlated, removing some does not have a large impact. This holds until the most correlated samples have been removed.

On other datasets, which have lower LD, perrformance falls at a more constant

rate although not entirely linearly. Once again, the Wheat dataset has the "noisier" results.

Model-wise, there are no significant differences. It seems that the results of these experiments seem to be driven not so much by model choice, but by the dataset and its nature. The links between these differences in performance and the trait, populations or relevant dataset characteristics require further examination.

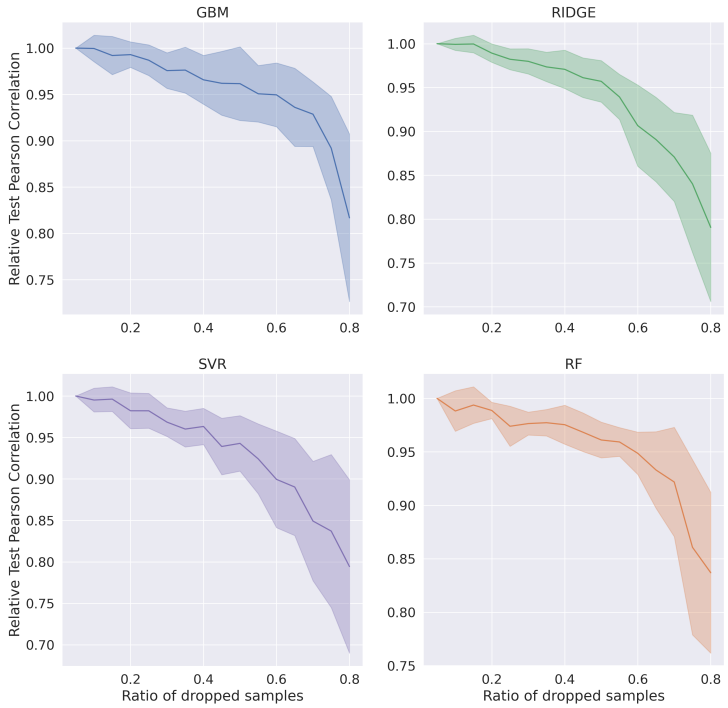


Figure F.4: Performance of different methods in the yeast dataset under varying amounts of dropped samples.

The decrease in predictive accuracy is significantly steeper when reducing the number of samples or adding phenotypic noise than in the case of marker elimination (see Section). Although the evidence is limited, this could point torwards a higher payoff for increasing the number of samples, or increasing the accuracy in phenotype measurements, than for increasing SNP marker density.

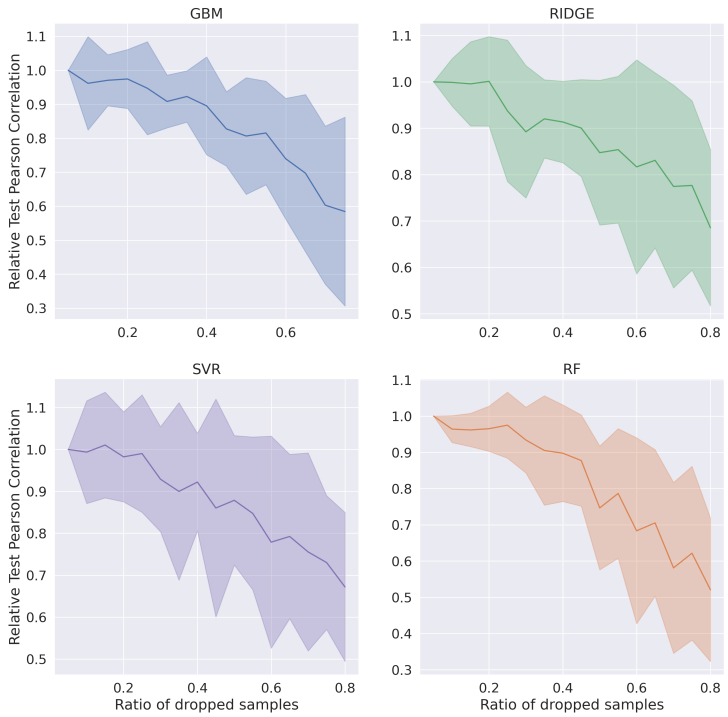


Figure F.5: Performance of different methods in the wheat dataset under varying amounts of dropped samples.

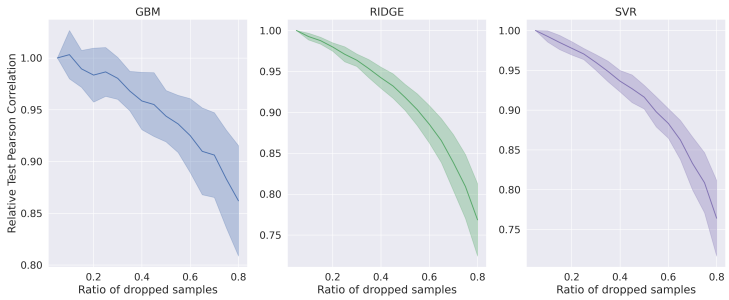


Figure F.6: Performance of different methods in the Holstein dataset under varying amounts of dropped samples.

Appendix G

Baseline Models Hyperparameters

.

The following tables show the hyperparameters obtained by carrying out a randomized search and cross-validation for different datasets, encodings and traits. This process was repeated for 10 train-test splits. For numerical parameters the mean and standard deviation across those 10 splits is included, and for categorical parameters the most frequent value is shown.

Although results vary across datasets, traits and environments some overall trends could be observed. Interestingly ridge penalty was higher for Additive encoding than for OHE, and largeIn this section we informally compare the parameters obtained to *default* ones on sklearn (Pedregosa et al., 2011). for both. The number of estimators and minimum number of samples per leaf in Random Forests and Gradient Boosting was also high (> 1000 and > 20, respectively). All these parameters have a regularising effect, and thus this result can be linked to the fact that targets are noisy, inputs high dimensional, and data scarce. However, in the case of SVR, C was not outstandingly large. In the case of SVR, radial basis functions showed the best performance on most -but not all - datasets and traits.

Table G.1: Hyper-parameter values for Support Vector Regression found using a randomized search and cross-validation.

Dataset	encoding	kernel	C mean	C std	epsilon mean	epsilon std
jersey	Add	rbf	1.253	0.171	0.346	0.126
holstein	OHE	rbf	1.934	0.000	0.100	0.000
holstein	Add	polynomial	0.952	0.000	0.121	0.000
wheat	Add	rbf	2.636	0.346	0.080	0.040
wheat	OHE	rbf	2.552	0.407	0.262	0.032
yeast	Add	linear	0.406	0.282	0.216	0.044

Table G.2: Hyper-parameter values for Gradient Boosting Machines found using a randomized search and cross-validation. For numerical parametters, the mean across splits is shown.

Dataset	encoding	max features	min impurity decrease	min samples leaf	subsample	n estimators	min samples split
jersey	Add	sqrt	0.781	60.000	0.867	1325.500	53.100
jersey	OHE	sqrt	0.756	34.000	0.734	1331.100	75.000
holstein	Add	sqrt	0.883	25.200	0.952	1586.700	30.700
holstein	OHE	sqrt	0.807	12.000	0.830	1763.000	24.000
wheat	OHE	sqrt	0.824	22.000	1.000	1564.900	41.300
wheat	Add	sqrt	0.571	39.000	1.000	1673.900	50.400
yeast	Add	sqrt	0.298	23.800	1.000	1385.100	29.000

Table G.3: Hyper-parameter values for Ridge found using a randomized search and cross-validation.

Dataset	encoding	trait/env	alpha mean	alpha std
holstein	Add	0	20000.000	0.000
holstein	Add	1	20000.000	0.000
holstein	Add	2	20000.000	0.000
holstein	OHE	0	10000.000	0.000
holstein	OHE	1	10000.000	0.000
holstein	OHE	2	10000.000	0.000
wheat	Add	1	80000.000	0.000
wheat	Add	2	80000.000	0.000
wheat	Add	3	80000.000	0.000
wheat	Add	0	20000.000	0.000
wheat	OHE	0	20000.000	0.000
wheat	OHE	1	20000.000	0.000
wheat	OHE	2	20000.000	0.000
wheat	OHE	3	20000.000	0.000
yeast	Add	0	80000.000	0.000
yeast	Add	1	80000.000	0.000
yeast	Add	2	80000.000	0.000
yeast	Add	3	80000.000	0.000
yeast	Add	4	80000.000	0.000
yeast	Add	5	80000.000	0.000
yeast	Add	6	80000.000	0.000
yeast	Add	7	80000.000	0.000
yeast	Add	8	80000.000	0.000
yeast	Add	9	80000.000	0.000
yeast	Add	10	80000.000	0.000
yeast	Add	11	80000.000	0.000

Table G.4: Hyper-parameter values for Random Forests found using a randomized search and cross-validation. For numerical parametters, the mean across splits is shown.

Dataset	encoding	trait/env	max features	min impurity decrease	min samples leaf	n estimators	min samples split
holstein	Add	0	sqrt	0.000	11.000	1314.000	73.000
holstein	Add	2	sqrt	0.000	28.000	1359.600	16.000
holstein	OHE	0	sqrt	0.000	39.000	1731.000	13.000
holstein	OHE	2	sqrt	0.000	39.000	1731.000	13.000
holstein	OHE	1	sqrt	0.000	39.000	1731.000	13.000
holstein	Add	1	sqrt	0.000	180.000	1428.000	46.000
wheat	Add	0	sqrt	0.000	20.000	1371.000	66.000
wheat	Add	1	sqrt	0.000	20.000	1371.000	66.000
wheat	Add	3	sqrt	0.000	20.000	1371.000	66.000
wheat	Add	2	sqrt	0.000	20.000	1371.000	66.000
wheat	OHE	2	sqrt	0.000	19.000	1314.000	40.000
wheat	OHE	0	sqrt	0.000	17.400	1314.000	46.600
wheat	OHE	1	sqrt	0.000	19.000	1314.000	40.000
wheat	OHE	3	sqrt	0.000	19.000	1314.000	40.000
yeast	Add	0	sqrt	0.055	13.800	1437.300	20.600
yeast	Add	1	sqrt	0.050	15.500	1524.900	17.000
yeast	Add	2	sqrt	0.050	20.800	1262.400	27.900
yeast	Add	3	sqrt	0.050	23.100	1320.500	24.600
yeast	Add	5	sqrt	0.050	21.800	1558.100	22.600
yeast	Add	4	sqrt	0.050	15.500	1524.900	17.000
yeast	Add	6	sqrt	0.050	16.700	1758.200	15.900
yeast	Add	7	sqrt	0.050	6.900	1495.800	12.800
yeast	Add	8	sqrt	0.050	21.100	1499.700	23.200
yeast	Add	9	sqrt	0.050	19.700	1516.500	19.200
yeast	Add	10	sqrt	0.050	23.500	1524.800	20.000
yeast	Add	11	sqrt	0.050	22.500	1370.700	24.700

References

Abdollahi-Arpanahi, R., Gianola, D., & Peñagaricano, F. (2020). Deep learning versus parametric and ensemble methods for genomic prediction of complex phenotypes. *Genetics Selection Evolution*, *52*(1), 1–15.

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. , 420–434.

Akdemir, D., & Isidro-Sánchez, J. (2019). Design of training populations for selective phenotyping in genomic prediction. *Scientific Reports*, *9*(1), 1446. Retrieved from `https://doi.org/10.1038/s41598-018-38081-6` doi: 10.1038/s41598-018-38081-6

Ando, R., & Zhang, T. (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *J. Mach. Learn. Res.*, *6*, 1817-1853.

Ashley-Koch, A., Yang, Q., & Olney, R. S. (2000, 05). Sickle Hemoglobin (Hb S) Allele and Sickle Cell Disease: A HuGE Review. *American Journal of Epidemiology*, *151*(9), 839-845. Retrieved from `https://doi.org/10.1093/oxfordjournals.aje.a010288` doi: 10.1093/oxfordjournals.aje.a010288

Azodi, C. B., McCarren, A., Roantree, M., de los Campos, G., & Shiu, S.-H. (2019). Benchmarking algorithms for genomic prediction of complex traits. *bioRxiv*, 614479.

Bartels, M. (2015). Genetics of wellbeing and its components satisfaction with life, happiness, and quality of life: A review and meta-analysis of heritability studies. *Behavior genetics*, *45*(2), 137–156.

Belda, J., Vergara, L., Safont, G., & Salazar, A. (2019). Computing the partial correlation of ica models for non-gaussian graph signal processing. *Entropy*, *21*(1), 22.

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, *1*(3), 81–84.

Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is "nearest neighbor" meaningful? , 217–235.

Biewald, L. (2020). *Experiment tracking with weights and biases.* Retrieved from `https://www.wandb.com/`

Bloom, J. S., Ehrenreich, I. M., Loo, W. T., Lite, T.-L. V., & Kruglyak,

L. (2013). Finding the sources of missing heritability in a yeast cross. *Nature*, *494*(7436), 234–237.

Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79.

Bohra, A., Chand Jha, U., Godwin, I. D., & Kumar Varshney, R. (2020). Genomic interventions for sustainable agriculture. *Plant Biotechnology Journal*, *18*(12), 2388–2405.

Botta, V., Louppe, G., Geurts, P., & Wehenkel, L. (2014). Exploiting snp correlations within random forest for genome-wide association studies. *PloS one*, *9*(4), e93379.

Brandes, U. (2005). *Network analysis: methodological foundations* (Vol. 3418). Springer Science & Business Media.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5-32.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., ... Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *Ecml pkdd workshop: Languages for data mining and machine learning* (pp. 108–122).

Caruana, R. (1997). Multitask learning. *Machine learning*, *28*(1), 41–75.

Cebamanos, L., Gray, A., Stewart, I., & Tenesa, A. (2014). Regional heritability advanced complex trait analysis for gpu and traditional parallel architectures. *Bioinformatics*, *30*(8), 1177–1179.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., ... Zecchina, R. (2019). Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, *2019*(12), 124018.

Chervonenkis, A. (2013, 10). Early history of support vector machines. In (p. 13-20). doi: 10.1007/978-3-642-41136-6_3

Chung, F. R., & Graham, F. C. (1997). *Spectral graph theory* (No. 92). American Mathematical Soc.

Clark, S. A., & van der Werf, J. (2013). Genomic best linear unbiased prediction (gblup) for the estimation of genomic breeding values. , 321–330.

Cohen, T. S., & Welling, M. (2016). Group equivariant convolutional networks. *CoRR*, *abs/1602.07576*. Retrieved from `http://arxiv.org/abs/1602.07576`

Covarrubias-Pazaran, G. (2016). Genome-assisted prediction of quantitative traits using the r package sommer. *PloS one*, *11*(6), e0156744.

Crossa, J., de Los Campos, G., Pérez, P., Gianola, D., Burgueño, J., Araus, J. L., ... others (2010). Prediction of genetic values of quantitative traits in plant breeding using pedigree and molecular markers. *Genetics*, *186*(2), 713–724.

Crossa, J., Perez, P., Hickey, J., Burgueno, J., Ornella, L., Cerón-Rojas, J.,

... others (2014). Genomic prediction in cimmyt maize and wheat breeding programs. *Heredity*, *112*(1), 48–60.

Dangalchev, C. (2006). Residual closeness in networks. *Physica A: Statistical Mechanics and its Applications*, *365*(2), 556–564.

de los Campos, G., Gianola, D., & Rosa, G. J. (2009). Reproducing kernel hilbert spaces regression: a general framework for genetic evaluation. *Journal of animal science*, *87*(6), 1883–1887.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

Dung, S. K., López, A., Barragan, E. L., Reyes, R.-J., Thu, R., Castellanos, E., ... Rohlfs, R. V. (2019). Illuminating women's hidden contribution to historical theoretical population genetics. *Genetics*, *211*(2), 363–366.

Dunn, J. (2016). Introducing fblearner flow: Facebook's ai backbone. *Engineering Blog, Facebook Code*.

Dziugaite, G. K., & Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*.

Endelman, J. B. (2011). Ridge regression and other kernels for genomic selection with r package rrblup. *The Plant Genome*, *4*(3), 250–255.

Evans, D. M., Visscher, P. M., & Wray, N. R. (2009). Harnessing the information contained within genome-wide association studies to improve individual prediction of complex disease risk. *Human molecular genetics*, *18*(18), 3525–3531.

Evgeniou, T., & Pontil, M. (2004). Regularized multi–task learning. In (p. 109–117). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/1014052.1014067` doi: 10.1145/1014052.1014067

Falconer, D. S., & Mackay, F. C. (1996). *Introduction to quantitative genetics*. Longman.

Fang, C., Shang, Y., & Xu, D. (2019). Prediction of protein backbone torsion angles using deep residual inception neural networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *16*(3), 1020-1028. doi: 10.1109/TCBB.2018.2814586

Fattahi, S., & Sojoudi, S. (2019). Graphical lasso and thresholding: Equivalence and closed-form solutions. *Journal of Machine Learning Research*, *20*(10), 1-44. Retrieved from `http://jmlr.org/papers/v20/17-501.html`

Figure8. (2018). *Data scientist report.* Retrieved from `https://visit.figure-eight.com/rs/416-ZBE-142/images/Data-Scientist-Report.pdf`

Fisher, R. A., et al. (1918). 009: The correlation between relatives on the supposition of mendelian inheritance.

Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, *5*(6), 345.

Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. (2020). Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*.

Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 35–41.

Friedman, J. H. (2001). Greedy function approximation: A gradient boostingmachine. *The Annals of Statistics*, *29*(5), 1189 – 1232.

Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, *5*(4), 322–333.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning* (pp. 1050–1059).

Gama, F., Isufi, E., Leus, G., & Ribeiro, A. (2020). Graphs, convolutions, and neural networks. *CoRR*, *abs/2003.03777*. Retrieved from `https://arxiv.org/abs/2003.03777`

Garofolo, J. S. (1993). Timit acoustic phonetic continuous speech corpus. *Linguistic Data Consortium, 1993*.

Gaudelet, T., Day, B., Jamasb, A. R., Soman, J., Regep, C., Liu, G., ... others (2020). Utilising graph machine learning within drug discovery and development. *arXiv preprint arXiv:2012.05716*.

Gerstein, M. B., Bruce, C., Rozowsky, J. S., Zheng, D., Du, J., Korbel, J. O., ... Snyder, M. (2007). What is a gene, post-encode? history and updated definition. *Genome research*, *17*(6), 669–681.

Gianola, D. (2013, 07). Priors in whole-genome regression: the bayesian alphabet returns. *Genetics*, *194*(3), 573–596. Retrieved from `https://pubmed.ncbi.nlm.nih.gov/23636739` doi: 10.1534/genetics.113.151753

Gianola, D., & Fernando, R. L. (1986). Bayesian methods in animal breeding theory. *Journal of Animal Science*, *63*(1), 217–244.

Gianola, D., & Van Kaam, J. B. (2008). Reproducing kernel hilbert spaces regression methods for genomic assisted prediction of quantitative traits. *Genetics*, *178*(4), 2289–2303.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning* (pp. 1263–1272).

Glorot, X., & Bengio, Y. (2010a). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Glorot, X., & Bengio, Y. (2010b, 13–15 May). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR. Retrieved from `http://proceedings.mlr.press/v9/glorot10a.html`,

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

Goldstein, B. A., Hubbard, A. E., Cutler, A., & Barcellos, L. F. (2010). An application of random forests to a genome-wide association dataset: methodological considerations & new findings. *BMC genetics*, *11*(1), 49.

González-Recio, O., Jiménez-Montero, J., & Alenda, R. (2013). The gradient boosting algorithm and random boosting for genome-assisted evaluation in large data sets. *Journal of dairy science*, *96*(1), 614–624.

González-Recio, O., Rosa, G. J., & Gianola, D. (2014). Machine learning methods and predictive ability metrics for genome-wide prediction of complex traits. *Livestock Science*, *166*, 217–231.

Griffiths, A. J. F., Wessler, S. R., Carroll, S. B., & Doebley, J. (2015). *Introduction to genetic analysis*. Eleventh edition. New York, NY : W.H. Freeman & Company. Retrieved from `https://search.library.wisc.edu/catalog/9910217031402121`

Grinberg, N. F., Orhobor, O. I., & King, R. D. (2018). An evaluation of machine-learning for predicting phenotype: studies in yeast, rice, and wheat. *Machine Learning*, 1–27.

Grinberg, N. F., Orhobor, O. I., & King, R. D. (2019). An evaluation of machine-learning for predicting phenotype: studies in yeast, rice, and wheat. *Machine Learning*, 1–27.

Groisman, P., Jonckheere, M., & Sapienza, F. (2018). Nonhomogeneous euclidean first-passage percolation and distance learning. *arXiv preprint arXiv:1810.09398*.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). *Deep residual learning for image recognition.*

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

In *Proceedings of the ieee international conference on computer vision* (pp. 1026–1034).

Heather, J. M., & Chain, B. (2016). The sequence of sequencers: The history of sequencing dna. *Genomics*, *107*(1), 1–8.

Heckel, R., Shomorony, I., Ramchandran, K., & David, N. (2017). Fundamental limits of dna storage systems. In *2017 ieee international symposium on information theory (isit)* (pp. 3130–3134).

Hermann, J., & Del Balso, M. (2017). Meet michelangelo: Uber's machine learning platform. *URL https://eng. uber. com/michelangelo*.

Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., ... Wilson, K. (2016). *Cnn architectures for large-scale audio classification.*

Hindorff, L. A., Sethupathy, P., Junkins, H. A., Ramos, E. M., Mehta, J. P., Collins, F. S., & Manolio, T. A. (2009). Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. *Proceedings of the National Academy of Sciences*, *106*(23), 9362–9367.

Hinton, G., & van Camp, D. (n.d.). Keeping neural networks simple by minimising the description length of weights. 1993. In *Proceedings of colt-93* (pp. 5–13).

Hochreiter, S., & Schmidhuber, J. (1995). Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems* (pp. 529–536).

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Holliday, J. A., Wang, T., & Aitken, S. (2012). Predicting adaptive phenotypes from multilocus genotypes in sitka spruce (picea sitchensis) using random forest. *G3: Genes— genomes— genetics*, *2*(9), 1085–1093.

Hooker, S. (2020). The hardware lottery. *arXiv preprint arXiv:2009.06489*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, *abs/1502.03167*. Retrieved from `http://arxiv.org/abs/1502.03167`

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., & Bengio, S. (2019). Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*.

Kaggle. (2019). *State of data science and machine learning.* Retrieved from `https://www.kaggle.com/kaggle-survey-2019`

Kazi, A., Cosmo, L., Navab, N., & Bronstein, M. (2020). Differentiable graph module (dgm) graph convolutional networks. *arXiv preprint arXiv:2002.04999*.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P.

(2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Kim, S. S., Dai, C., Hormozdiari, F., van de Geijn, B., Gazal, S., Park, Y., ... others (2019). Genes with high network connectivity are enriched for disease heritability. *The American Journal of Human Genetics*, *104*(5), 896–913.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kolaczyk, E. D., & Csárdi, G. (2020). Descriptive analysis of network graph characteristics. In *Statistical analysis of network data with r* (pp. 43–68). Springer.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th international conference on neural information processing systems - volume 1* (p. 1097–1105). Red Hook, NY, USA: Curran Associates Inc.

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PloS one*, *12*(5), e0177459.

Lang, K. (1995). Newsweeder: Learning to filter netnews. , 331–339.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989, 12). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541-551. Retrieved from `https://doi.org/10.1162/neco.1989.1.4.541` doi: 10.1162/neco.1989.1.4.541

Lee, T., & Lee, I. (2018). aragwab: network-based boosting of genome-wide association studies in arabidopsis thaliana. *Scientific reports*, *8*(1), 1–6.

Leiserson, C. E., Thompson, N. C., Emer, J. S., Kuszmaul, B. C., Lampson, B. W., Sanchez, D., & Schardl, T. B. (2020). There's plenty of room at the top: What will drive computer performance after moore's law? *Science*, *368*(6495).

Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*.

Liu, Y., Wang, D., He, F., Wang, J., Joshi, T., & Xu, D. (2019). Phenotype prediction and genome-wide association study using deep convolutional neural network of soybean. *Frontiers in genetics*, *10*, 1091.

Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2019). Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*.

Lutzeyer, J., & Walden, A. (2017). Comparing graph spectra of adjacency

and laplacian matrices. *arXiv preprint arXiv:1712.03769*.

Ma, S., Bassily, R., & Belkin, M. (2018). The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International conference on machine learning* (pp. 3325–3334).

Ma, W., Qiu, Z., Song, J., Cheng, Q., & Ma, C. (2017). Deepgs: Predicting phenotypes from genotypes using deep learning. *bioRxiv*, 241414.

Ma, W., Qiu, Z., Song, J., Li, J., Cheng, Q., Zhai, J., & Ma, C. (2018). A deep convolutional neural network approach for predicting phenotypes from genotypes. *Planta*, *248*(5), 1307–1318.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml* (Vol. 30, p. 3).

Meghanathan, N. (2014). Spectral radius as a measure of variation in node degree for complex network graphs. In *2014 7th international conference on u-and e-service, science and technology* (pp. 30–33).

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, *2014*(239), 2.

Mianjy, P., Arora, R., & Vidal, R. (2018). On the implicit bias of dropout. In *International conference on machine learning* (pp. 3540–3548).

Mohar, B., Alavi, Y., Chartrand, G., & Oellermann, O. (1991). The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, *2*(871-898), 12.

Morota, G., & Gianola, D. (2014). Kernel-based whole-genome prediction of complex traits: a review. *Frontiers in Genetics*, *5*, 363. Retrieved from `https://www.frontiersin.org/article/10.3389/fgene.2014.00363` doi: 10.3389/fgene.2014.00363

Morota, G., Koyama, M., Rosa, G. J., Weigel, K. A., & Gianola, D. (2013). Predicting complex traits using a diffusion kernel on genetic markers with an application to dairy cattle and wheat data. *Genetics Selection Evolution*, *45*(1), 17.

Mourad, R., Sinoquet, C., & Leray, P. (2011, 03). Probabilistic graphical models for genetic association studies. *Briefings in Bioinformatics*, *13*(1), 20-33. Retrieved from `https://doi.org/10.1093/bib/bbr015` doi: 10.1093/bib/bbr015

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2019). Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*.

Nesmachnow, S., & Iturriaga, S. (2019). Cluster-uy: Collaborative scientific high performance computing in uruguay. In M. Torres & J. Klapp (Eds.), *Supercomputing* (pp. 188–202). Cham: Springer International Publishing.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr* (Vol. 269, pp. 543–547).

O'brien, M., Costin, B., & Miles, M. (2012). Using genome-wide expression profiling to define gene networks relevant to the study of complex traits: from rna integrity to network topology. *International review of neurobiology*, *104*, 91–133.

Palladino, P. (1993). Between craft and science: plant breeding, mendelian genetics, and british universities, 1900-1920. *Technology and Culture*, *34*(2), 300–323.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peixoto, T. P. (2014). The graph-tool python library. *figshare*. Retrieved 2014-09-10, from `http://figshare.com/articles/graph_tool/1164194` doi: 10.6084/m9.figshare.1164194

Pérez, P., & de Los Campos, G. (2014). Genome-wide regression and prediction with the bglr statistical package. *Genetics*, *198*(2), 483–495.

Pérez-Enciso, M., & Zingaretti, L. M. (2019). A guide on deep learning for complex trait genomic prediction. *Genes*, *10*(7), 553.

Philipp, G., Song, D., & Carbonell, J. G. (2017). The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv preprint arXiv:1712.05577*.

Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M. A., Bender, D., ... others (2007). Plink: a tool set for whole-genome association and population-based linkage analyses. *The American journal of human genetics*, *81*(3), 559–575.

Qanbari, S., Gianola, D., Hayes, B., Schenkel, F., Miller, S., Moore, S., ... Simianer, H. (2011). Application of site and haplotype-frequency based approaches for detecting selection signatures in cattle. *BMC genomics*, *12*(1), 318.

R Core Team. (2020). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from `https://www.R-project.org/`

Ramsey, J. D. (2014). A scalable conditional independence test for nonlinear, non-gaussian data. *arXiv preprint arXiv:1401.5031*.

Rezende, F. M., Nani, J. P., & Peñagaricano, F. (2019). Genomic prediction of bull fertility in us jersey dairy cattle. *Journal of dairy science*, *102*(4), 3230–3240.

Ribeiro, A. (2020a). `https://gnn.seas.upenn.edu/wp-content/uploads/2020/09/lecture_3_handout.pdf`.

Ribeiro, A. (2020b). `https://gnn.seas.upenn.edu/wp-content/uploads/`

2020/09/lecture_4_handout.pdf.

Ritchie, M. D. (2012). The success of pharmacogenomics in moving genetic association studies from bench to bedside: study design and implementation of precision medicine in the post-gwas era. *Human genetics*, *131*(10), 1615–1626.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Rochat, Y. (2009). *Closeness centrality extended to unconnected graphs: The harmonic centrality index* (Tech. Rep.).

Rohde, P. D., Fourie Sørensen, I., & Sørensen, P. (2020). qgg: an r package for large-scale quantitative genetic analyses. *Bioinformatics*, *36*(8), 2614–2615.

Rosa, G. J., Felipe, V. P., & Peñagaricano, F. (2016). Applications of graphical models in quantitative genetics and genomics. In *Systems biology in animal production and health, vol. 1* (pp. 95–116). Springer.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2019). *How does batch normalization help optimization?*

Sapienza, F., Groisman, P., & Jonckheere, M. (2018). Weighted geodesic distance following fermat's principle.

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, *10*(5), 1299–1319.

Segarra, S., Huang, W., & Ribeiro, A. (2019). *Lecture handout: Signal processing on graphs.* https://www.seas.upenn.edu/~ese224/slides/900_graph_signal_processing.pdf. (Accessed: 2020–11-06)

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Sharma, A., Vans, E., Shigemizu, D., Boroevich, K. A., & Tsunoda, T. (2019). Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific reports*, *9*(1), 1–7.

Sinoquet, C. (2014). *Probabilistic graphical models for genetics, genomics, and postgenomics*. OUP Oxford.

Sorensen, D., & Gianola, D. (2007). *Likelihood, bayesian, and mcmc methods in quantitative genetics*. Springer Science & Business Media.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929-1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html

Sutton, R. (2019). *The bitter lesson.* Retrieved from http://www

.incompleteideas.net/IncIdeas/BitterLesson.html

Sved, J. A., McRae, A. F., & Visscher, P. M. (2008). Divergence between human populations estimated from linkage disequilibrium. *The American Journal of Human Genetics*, *83*(6), 737–743.

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, *abs/1905.11946*. Retrieved from http://arxiv.org/abs/1905.11946

Theunissen, B. (2008). Breeding without mendelism: theory and practice of dairy cattle breeding in the netherlands 1900–1950. *Journal of the History of Biology*, *41*(4), 637–676.

Thieffry, D., Huerta, A. M., Pérez-Rueda, E., & Collado-Vides, J. (1998). From specific gene regulation to genomic networks: a global analysis of transcriptional regulation in escherichia coli. *Bioessays*, *20*(5), 433–440.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, *58*, 267-288.

Tsalenko, A., Sharan, R., Kristensen, V., Edvardsen, H., Børresen-Dale, A.-L., Ben-Dor, A., & Yakhini, Z. (2006). Analysis of snp-expression association matrices. *Journal of bioinformatics and computational biology*, *4*(02), 259–274.

Tusell, L., Pérez-Rodríguez, P., Forni, S., Wu, X., & Gianola, D. (2013). Genome-enabled methods for predicting litter size in pigs: a comparison. *Animal: an international journal of animal bioscience*, *7*(11), 1739.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(11).

VanRaden, P. M. (2008). Efficient methods to compute genomic predictions. *Journal of dairy science*, *91*(11), 4414–4423.

Vapnik, V. (1991). Principles of risk minimization for learning theory.

Vapnik, V. (1995). *The nature of statistical learning theory*. Berlin, Heidelberg: Springer-Verlag.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Vazquez, A., Bates, D., Rosa, G., Gianola, D., & Weigel, K. (2010). an r package for fitting generalized linear mixed models in animal breeding. *Journal of animal science*, *88*(2), 497–504.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Waldmann, P. (2019). On the use of the pearson correlation coefficient for model evaluation in genome-wide prediction. *Frontiers in genetics*, *10*, 899.

Wang, D., Zeng, S., Xu, C., Qiu, W., Liang, Y., Joshi, T., & Xu, D. (2017, 08). MusiteDeep: a deep-learning framework for general and kinase-specific phosphorylation site prediction. *Bioinformatics*, *33*(24), 3909-3916. Retrieved from https://doi.org/10.1093/bioinformatics/btx496 doi: 10.1093/bioinformatics/btx496

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, *38*(5), 1–12.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, *393*(6684), 440–442.

Wei, C., & Ma, T. (2019). Improved sample complexities for deep networks and robust classification via an all-layer margin. *arXiv preprint arXiv:1910.04284*.

Weigel, K. A., VanRaden, P. M., Norman, H. D., & Grosu, H. (2017, 2020/03/16). A 100-year review: Methods and impact of genetic selection in dairy cattle. from daughter dam comparisons to deep learning algorithms. *Journal of Dairy Science*, *100*(12), 10234–10250. Retrieved from https://doi.org/10.3168/jds.2017-12954 doi: 10.3168/jds.2017-12954

Wen, Z., Shi, J., Li, Q., He, B., & Chen, J. (2018). Thundersvm: A fast svm library on gpus and cpus. *The Journal of Machine Learning Research*, *19*(1), 797–801.

Weng, L. (2018). *Attention? attention!* Retrieved from https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Wolak, M. E. (2012). nadiv: an r package to create relatedness matrices for estimating non-additive genetic variances in animal models. *Methods in Ecology and Evolution*, *3*(5), 792–796.

Wray, N. R., Yang, J., Hayes, B. J., Price, A. L., Goddard, M. E., & Visscher, P. M. (2013). Pitfalls of predicting complex traits from snps. *Nature Reviews Genetics*, *14*(7), 507–515.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020a). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020b). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1492–1500).

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evalua-

tion of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Yin, L., Zhang, H., Zhou, X., Yuan, X., Zhao, S., Li, X., & Liu, X. (2020). Kaml: improving genomic prediction accuracy of complex traits using machine learning determined parameters. *Genome biology*, *21*(1), 1–22.

Yoo, A. B., Jette, M. A., & Grondona, M. (2003). Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing* (pp. 44–60).

Zaabza, H. B., Gara, A. B., & Rekik, B. (2017). Bayesian modeling in genetics and genomics. *Bayesian Inference*, 207.

Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, *33*(4), 452–473.

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... others (2018). Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, *41*(4), 39–45.

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Zhang, L., & Kim, S. (2014). Learning gene networks under snp perturbations using eqtl datasets. *PLoS Comput Biol*, *10*(2), e1003420.

Zhang, Z., Erbe, M., He, J., Ober, U., Gao, N., Zhang, H., ... Li, J. (2015). Accuracy of whole-genome prediction using a genetic architecture-enhanced variance-covariance matrix. *G3: Genes, Genomes, Genetics*, *5*(4), 615–627.

Zheng, X., Levine, D., Shen, J., Gogarten, S. M., Laurie, C., & Weir, B. S. (2012). A high-performance computing toolset for relatedness and principal component analysis of snp data. *Bioinformatics*, *28*(24), 3326–3328.

Zou, H., & Hastie, T. (2003). Regression shrinkage and selection via the elastic net, with applications to microarrays. *JR Stat Soc Ser B*, *67*, 301–20.

This page intentionally left blank.

List of Tables

List of Figures

This is the last page.
Compiled Monday 31st May, 2021.
http://iie.fing.edu.uy/