Osval Antonio Montesinos López
Abelardo Montesinos López
José Crossa

Multivariate Statistical Machine Learning Methods for Genomic Prediction

Foreword by Fred van Eeuwijk

OPEN ACCESS

Springer

Multivariate Statistical Machine Learning Methods for Genomic Prediction

Osval Antonio Montesinos López •
Abelardo Montesinos López • José Crossa

Multivariate Statistical Machine Learning Methods for Genomic Prediction

Springer

Foreword by
Fred van Euwijk

Osval Antonio Montesinos López
Facultad de Telemática
University of Colima
Colima, México

Abelardo Montesinos López
Departamento de Matemáticas
University of Guadalajara
Guadalajara, México

José Crossa
Biometrics and Statistics Unit
CIMMYT
Texcoco, Estado de México, México

Colegio de Postgraduados
Montecillos, Estado de México, México

ISBN 978-3-030-89009-4 ISBN 978-3-030-89010-0 (eBook)
https://doi.org/10.1007/978-3-030-89010-0

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

"This book is a prime example of CIMMYT's commitment to scientific advancement, and comprehensive and inclusive knowledge sharing and dissemination. The book presents novel models and methods for genomic selection theory and aims to facilitate their adoption and use by publicly funded researchers and practitioners of National Agricultural Research Extension Systems (NARES) and universities across the Global South. The objectives of the authors is to offer an exhaustive overview of the current state of the art to give access to the different new models, methods, and techniques available to breeders who often struggle with limited resources and/or practical constraints when implementing genomics-assisted selection. This aspiration would not be possible without the continuous support of CIMMYT's partners and donors who have funded non-profit frontier research for the benefit of millions of farmers and low-income communities worldwide. In this regard, it could not be more fitting for this book to be published as an open access resource for the international plant breeding community to benefit from. I trust that this publication will become a mandatory reference in the field of genomics-assisted breeding and that it will greatly contribute to accelerate the development and deployment of resource efficient and nutritious crops for a hotter and drier world."

Bram Govaerts, Director General, a.i.
CIMMYT

Foreword

In the field of plant breeding, the concept of prediction has always played a key role. For a long time, breeders tried to predict superior genetic performance, genotypic value or breeding value, depending on the mating system of the crop, from observations on a phenotype of interest with estimators that relied on the ratio of genetic to phenotypic variance. This ratio was usually the heritability on a line mean basis. When molecular markers became available, around 1990, it did not take too long before predictions of genotypic values were created from linear mixed models containing design or predictor matrices built on marker information. In plants, a paper by Rex Bernardo in 1994 (*Crop Science*, 34) was probably the first to propose this. A similar idea, under the name of genomic prediction and selection, revolutionized animal breeding after the publication of a famous paper by Theo Meuwissen, Ben Hayes, and Mike Goddard in *Genetics* 157, in 2001.

In contrast to its immediate success in animal breeding, in plant breeding the introduction and absorption of genomic prediction took longer. A reason for this difference between animal and plant breeding can be that for many traits in plant breeding it was not that difficult to identify useful quantitative trait loci (QTLs), whereas in animal breeding the detection of QTLs was more cumbersome. Therefore, in plant breeding some progress was possible by manipulating QTL alleles in marker-assisted selection procedures. Another reason for the slower implementation of genomic prediction in plants is the smaller order of magnitude for the training sets. In animal breeding 1000s or 10,000s are common; in plants 1000 is an outlier at the high end. A further and major complication in genomic prediction procedures for plants is the ubiquity of genotype by environment interaction.

Many methods have been proposed for genomic prediction over the last 20 years. Because these methods aim at using large numbers of SNPs (or other markers/sequence information) for prediction of genotypic values, the number of predictors is typically higher, and often a lot higher, than the number of genotypes with phenotypes, i.e., the size of the training set. Therefore, some form of regularization is necessary in the estimation procedure for the SNP effects. This regularization can come in ways familiar to plant breeders. The classical linear mixed model for

prediction of genotypic values, which uses line mean heritability and random genotypic effects that are structured by pedigree relations, can be generalized to a similar linear mixed with genotypic effects structured by marker relationships between genotypes. In that case, we move from classical best linear unbiased predictions, or BLUPs, to genome-enabled BLUPs, or G-BLUPs.

The G-BLUP has become the workhorse of genomic prediction in animal and plant breeding. For computational reasons, whenever the number of genotypes is less than the number of markers, a mixed model formulation that structures relationships between genotypes from marker profiles is more attractive than a formulation inserting a design matrix containing the SNPs itself. The latter formulation is a form of Ridge regression, and its predictions are called RR-BLUPs. G-BLUP and RR-BLUP can be motivated from a shrinkage perspective within a mixed model context, but alternatively, motivations from a penalized regression or Bayesian context are possible. RR-BLUP and G-BLUP use a Ridge penalty, while they can also be understood from the use of a single Normal prior on the SNP effects. At this point, it is obvious that all kinds of other penalties and priors have been proposed in genomic prediction, with the idea of inserting information on the genetic architecture of the trait. For example, penalties and priors can induce a degree of sparsity, like in Lasso-type penalties and Laplace/double exponential priors. Further, different assumptions can be made on the distribution of the (non-null) SNP effects. In another direction, SNP effects can be assigned to belong to different distributions with different variances, i.e., different magnitudes of SNP effects.

The tool kit for predicting genotypic values, or better, expected phenotypic performance from marker profiles, was quickly filled with linear mixed models (G-BLUP, RR-BLUP), penalized regressions (Ridge, Lasso), and Bayesian methods (A, B, Cπ, R). An excellent review of all these methods was produced by de los Campos et al. in *Genetics* 193, 2013. In addition to the above "statistical" prediction methods, all kinds of machine learning techniques like support vector machines, random forests, and deep learners are proposed for genomic prediction. A good review and comparison of statistical and machine learning methods was offered by Azodi et al., *G3*, 2019. A somewhat reassuring and equally disappointing conclusion from this paper and many other papers on comparison of genomic prediction methods is that prediction accuracies do not change very much between methods.

The authors of the book that you have in front of you now played an important role in the evaluation of a wide spectrum of prediction methods in plants, starting from G-BLUP and extending to deep learning. Their work included real wheat and maize data from the well-known international breeding institute CIMMYT, the international center for the improvement of maize and wheat. They were also instrumental in generalizing genomic prediction methods to multiple traits and environments. The route chosen here was via a generalization of multi-trait and multi-environment mixed models by defining a structure for the random genotype $\times$ trait effects and genotype $\times$ environment effects as a Kronecker product of structuring matrices on genotypes on the one hand and traits or environments on the

other hand. For the genotypes, the relationships were defined by markers; for the traits or environments, unstructured or factor analytic variance-covariance matrices can be chosen. When explicit environmental characterizations are available, relationships between environments can also be based on environmental similarities, as explained by Jarquin et al. in *Theoretical and Applied Genetics* 127, 2014. As another major contribution, the book authors showed how to do genomic prediction for binary traits, categorical traits, and count data. The way forward for such data is via the use of Bayesian implementations of generalized linear (mixed) models.

A remarkable achievement of the current book is that the authors have been able to present a full spectrum of genomic prediction methods for plant data. The book gives the theory and practice of genomic predictions for single and multiple environments, single and multiple traits, for continuous data as well as for binary, categorical, and count data. For the "traditional statistical methods," mixed models are explained, alongside penalized regressions and Bayesian methods. For the machine learning techniques, we find kernel methods, support vector machines, random forests, and all kinds of old and modern machine and deep learners. In between we see functional regression. The theoretical treatments of the methods are rather advanced for average plant breeders, but the book simultaneously offers a lot of introductory material on the philosophy behind statistical learning, bias-variance trade-offs, training and test sets, cross validation, matrices, penalties, priors, kernels, trees, and deep learners that allows less statistically trained readers to get some flavor of most of the techniques. Another positive point on the usability of the book is the ample inclusion of R programs to do the analyses yourself. The central packages for prediction are BGLR, BMTME, glmnet, and Keras, with the first two closely connected to the authors of this book. Some worry may be expressed on the computational feasibility of the Bayesian implementations for larger data sets, i.e., 100s of genotypes and 30 or more traits/trials.

The book offers more than just methods for genomic prediction. Most of the prediction methods can easily be adapted to other types of prediction, where the target traits remain the same, while the inputs are not SNPs, but metabolites, proteins, gene expressions, methylations, and further omics data. Different types of predictors can also be combined as in kernel methods, where different types of inputs serve to define multiple kernels, which translate to (transformed) genotype $\times$ genotype similarity matrices that structure genotypic random effects. For example, two random genotypic effects may be defined, with a SNP-based relationship matrix structuring the first random genotypic effect and a metabolite-based relationship matrix structuring the second genotypic effect. Another example consists in adding a dominance kernel to a mixed model that already contains a kernel for the additive effects. To include epistatic effects of all orders in addition to additive effects, Reproducing Kernel Hilbert Space models can be used. For the use of high-throughput phenotyping information as predictors for phenotypic traits, functional regression methods are described. The use of images for prediction of plant traits is facilitated by deep learners.

This book is encyclopedic in its approach to prediction in plant breeding. About any feasible method is described with its statistical properties, examples, and

annotated software. Most if not all prevailing applications are covered. Readability and usefulness are improved by the fact that chapters are to a large extent self-contained. The book is a unique reference manual for people with an academic or professional interest in prediction methods in plants.

Wageningen, The Netherlands Fred van Eeuwijk
1 September 2021

Preface

In plant breeding, prediction models are essential for the successful implementation of genomic selection (GS), which is considered a predictive methodology used to train models with a reference training population containing known phenotypic and genotypic data and then perform predictions for a testing data set that only contains genomic data. However, because a universal model is nonexistent (free lunch theorem), it is necessary to evaluate many models for a particular data set and subsequently choose the best option for each particular situation. Thus, a great variety of statistical models are used for prediction in GS. The GS genome-based statistical machine prediction models face several complexities, as the choice of the model depends on multiple factors such as the genetic architecture of the trait, marker density, sample size, the span of linkage disequilibrium, and the genotype × environment interaction.

In the field of plant science, the most popular statistical learning model is the linear mixed model, which uses Henderson's equations to find the best linear unbiased estimates (BLUEs) for fixed effects, the best linear unbiased predictors (BLUPs) for random effects, and the Bayesian counterpart of this model from which the authors developed different versions that they coined into the so-called Bayesian Alphabet Methods [Bayesian Ridge Regression (BRR), BayesA, BayesB, BayesC, Bayes Lasso (BL), etc.]. Several machine learning methods are available, including deep learning, random forest, and support vector machines. Random forest and support vector machines are very easy to implement since they require few hyperparameters to be tuned and quickly provide very competitive predictions.

The methods arising from statistical and machine learning fields are called statistical machine learning methods.

All of the statistical machine learning methods require highly preprocessed inputs (feature engineering) to produce reasonable predictions. In other words, most statistical machine learning methods need more user intervention to preprocess inputs, which must be done manually. Chapter 1 assesses the use of big data for prediction and estimation through statistical machine learning and its applications in agriculture and genetics in general, and it provides key elements of genomic selection and its

potential for plant improvement. Chapter 2 details the data preparation necessary for implementing statistical machine learning methods for genomic selection. We present tools for cleaning, imputing, and detecting minor and major allele frequency computation, marker recodification, marker frequency of heterogeneous, frequency of NAs, and methods for computing the genomic relationship matrix. Chapter 3 provides details of the linear multiple regression model with gradient descendent methods described for learning the parameters under this model. Penalized linear multiple regression is derived for Ridge and Lasso penalties.

This book provides several examples in every chapter, including the necessary R codes, thus facilitating rapid comprehension and use.

Chapter 4 describes the overfitting phenomenon that occurs when a statistical machine learning model thoroughly learns the noise as well as the signal that is present in the training data, and the underfitted model when only a few predictors are included in the statistical machine learning model that represents the complete structure of the data pattern poorly. In Chapter 5, we explain the linear mixed model framework by exploring three methods of parameter estimation (maximum likelihood, EM algorithm and REML) and illustrate how genomic-enabled predictions are performed under this framework. The use of linear mixed models in Chap. 5 is illustrated for single-trait and multi-trait linear mixed models, and the R codes for performing the analyses are provided.

Chapter 6 describes the Bayesian paradigm for parameter estimation of several Bayesian linear regressions used in genomic-enabled prediction and present in various examples of predictors implemented in the Bayesian Generalized Linear Regression (BGLR) library for continuous response variables when including main effects (of environments and genotypes) as well as interaction terms related to genotype-environment interaction. The classic and Bayesian prediction paradigms for categorical and count data are delineated in Chap. 7; examples were implemented in the BGLR and glmnet library, where penalized multinomial logistic regression and penalized Poisson regression are given.

A notable semiparametric machine learning model is introduced in Chap. 8, where some loss functions under a fixed model framework with examples of Gaussian, binary, and categorical response variables are provided. We illustrate the use of mixed models with kernels including examples for multi-trait RKHS regression methods for continuous response variables.

We point out the origin and popularity of support vector machine methods in Chap. 9 and highlight their derivations by the maximum margin classifier and the support vector classifier, explaining the fundamentals for building the different kernel methods that are allowed in the support vector machine. In Chap. 10, we go through the foundations of artificial neural networks and deep learning methods, drawing attention to the main inspirations for how the methods of deep learning are built by defining the activation function and its role in capturing nonlinear patterns in the input data. Chapter 11 presents elements for implementing deep neural networks (deep learning, DL) for continuous outcomes. Several practical examples with plant breeding data for implementing deep neural networks in the keras library are outlined. The efficiency of artificial neural networks and deep learning methods is

extended to genome-based prediction in keras for binary, categorical, and mixed outcomes under feed-forward networks (Chap. 12). Deep learning is not frequently used in genome-based prediction, as its superiority in predicting performance of unobserved individuals over conventional statistical machine learning methods is still unclear. Nevertheless, few applications of DL for genomic-enabled prediction show that it improves the selection of candidate genotypes at an early stage in the breeding programs, as well as improving the understanding of the complex biological processes involved in the relationship between phenotypes and genotypes.

We provide several examples from plant breeding experiments including genomic data. Chapter 13 presents a formal motivation for Convolution Neural Networks (CNNs) that shows the advantages of this topology compared to feed-forward networks for processing images. Several practical examples with plant breeding data are provided using CNNs under two scenarios: (a) one-dimensional input data and (b) two-dimensional input data. We also give examples illustrating how to tune the hyperparameters in the CNNs.

Modern phenomics methods are able to use hyperspectral cameras to provide hundreds of reflectance data points at discrete narrow bands in many environments and at many stages of crop development. Phenotyping technology can now be used to quickly and accurately obtain data on agronomic traits based on extensive investments and advancements in plant phenotyping technologies. The main objective of HTP is to reduce the cost of data per plot and increase genomic-enabled prediction accuracy early in the crop-growing season. Chapter 14 presents the theoretical fundamentals and practical issues of using functional regression in the context of phenomic (images) and genomic-(dense molecular markers) enabled predictions. Functional regression represents data by means of basis functions. We derived a Bayesian version of functional regression and explain details of its implementation.

Chapter 15 provides a description of the random forest algorithm, the main hyperparameters that need to be tuned, as well as the different splitting rules that are key for its implementation. Several examples are provided for training random forest models with different types of response variables using plant breeding and genome-based prediction. A random forest algorithm for multivariate outcomes is provided, and its most popular splitting rules are also explained.

Colima, México — Osval Antonio Montesinos López
Guadalajara, México — Abelardo Montesinos López
Texcoco, Estado de México, México — José Crossa

Acknowledgments

The work described in this book is the result of more than 20 years of research on statistical and quantitative genetic sciences for developing models and methods for improving genome-enabled predictions. Most of the data used in this study was produced and provided by CIMMYT and diverse research partners. We are inspired and grateful for their tremendous commitment to advancing and applying science for public good and specially for improvising the livelihood of the low resource farmers.

The authors are grateful to the past and present CIMMYT Directors General, Deputy Directors of Research, Deputy Directors of Administration, Directors of Research Programs, and other Administration offices and Laboratories of CIMMYT for their continuous and firm support of biometrical genetics, and statistics research, training, and service in support of CIMMYT's mission: "maize and wheat science for improved livelihoods."

This work was made possible with support from the CGIAR Research Programs on Wheat and Maize (wheat.org, maize.org), and many funders including Australia, United Kingdom (DFID), USA (USAID), South Africa, China, Mexico (SAGARPA), Canada, India, Korea, Norway, Switzerland, France, Japan, New Zealand, Sweden, and the World Bank. We thank the financial support of the Mexico Government through out MASAGRO and several other regional projects and close collaboration with numerous Mexican researchers.

We acknowledge the financial support provided by the (1) Bill and Melinda Gates Foundation (INV-003439 BMGF/FCDO Accelerating Genetic Gains in Maize and Wheat for Improved Livelihoods [AG2MW]) as well as (2) USAID projects (Amend. No. 9 MTO 069033, USAID-CIMMYT Wheat/AGGMW, AGG-Maize Supplementary Project, AGG [Stress Tolerant Maize for Africa]).

Very special recognition is given to Bill and Melinda Gates Foundation for providing the Open Access fee of this book.

We are also thankful for the financial support provided by the (1) Foundations for Research Levy on Agricultural Products (FFL) and the Agricultural Agreement Research Fund (JA) in Norway through NFR grant 267806, (2) Sveriges Llantbruksuniversitet (Swedish University of Agricultural Sciences) Department of

Plant Breeding, Sundsvägen 10, 23053 Alnarp, Sweden, (3) CIMMYT CRP, (4) the Consejo Nacional de Tecnología y Ciencia (CONACYT) of México, (5) Universidad de Colima of Mexico, and (6) Universidad de Guadalajara of Mexico.

We highly appreciate and thank the several students at the Universidad de Colima, students and professors from the Colegio de Post-Graduados and the Universidad de Guadalajara who tested and made suggestions on early version of the material covered in the book; their many useful suggestions had a direct impact on the current organization and the technical content of the book.

Finally, we wish to express our deep thanks to Prof. Dr. Fred van Eeuwijk, who carefully read, corrected, modified, and suggested changes for each of the chapter of this book. We introduced many important suggestions and additions based on Prof. Dr. Fred van Eeuwijk's extensive scientific knowledge.

Contents

Chapter 1
General Elements of Genomic Selection and Statistical Learning

1.1 Data as a Powerful Weapon

Thanks to advances in digital technologies like electronic devices and networks, it is possible to automatize and digitalize many jobs, processes, and services, which are generating huge quantities of data. These "big data" are transmitted, collected, aggregated, and analyzed to deliver deep insights into processes and human behavior. For this reason, data are called the new oil, since "data are to this century what oil was for the last century"—that is, a driver for change, growth, and success. While statistical and machine learning algorithms extract information from raw data, information can be used to create knowledge, knowledge leads to understanding, and understanding leads to wisdom (Sejnowski 2018). We have the tools and expertise to collect data from diverse sources and in any format, which is the cornerstone of a modern data strategy that can unleash the power of artificial intelligence. Every single day we are creating around 2.5 quintillion bytes of data (McKinsey Global Institute 2016). This means that almost 90% of the data in the world has been generated over the last 2 years. This unprecedented capacity to generate data has increased connectivity and global data flows through numerous sources like tweets, YouTube, blogs, sensors, internet, Google, emails, pictures, etc. For example, Google processes more than 40,000 searches every second (and 3.5 billion searches per day), 456,000 tweets are sent, and 4,146,600 YouTube videos are watched per minute, and every minute, 154,200 Skype calls are made, 156 million emails are sent, 16 million text messages are written, etc. In other words, the amount of data is becoming bigger and bigger (big data) day by day in terms of volume, velocity, variety, veracity, and "value."

The nature of international trade is being radically transformed by global data flows, which are creating new opportunities for businesses to participate in the global economy. The following are some ways that these data flows are transforming international trade: (a) businesses can use the internet (i.e., digital platforms) to export goods; (b) services can be purchased and consumed online; (c) data collection

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_1

and analysis are allowing new services (often also provided online) to add value to exported goods; and (d) global data flows underpin global value chains, creating new opportunities for participation.

According to estimates, by 2020, 15–20% of the gross domestic product (GDP) of countries all over the world will be based on data flows. Companies that adopt big data practices are expected to increase their productivity by 5–10% compared to companies that do not, and big data practices could add 1.9% to Europe's GDP between 2014 and 2020. According to McKinsey Global Institute (2016) estimates, big data could generate an additional $3 trillion in value every year in some industries. Of this, $1.3 trillion would benefit the United States. Although these benefits do not directly affect the GDP or people's personal income, they indirectly help to improve the quality of life.

But once we have collected a large amount of data, the next question is: how to make sense of it? A lot of businesses and people are collecting data, but few are extracting useful knowledge from them. As mentioned above, nowadays there have been significant advances for measuring and collecting data; however, obtaining knowledge from (making sense of) these collected data is still very hard and challenging, since there are few people in the market with the expertise and training needed to extract knowledge from huge amounts of data. For this reason, to make sense of data, new disciplines were recently created, such as data science (the commercial name of statistics), business intelligence and analytics, that use a combination of statistics, computing, machine learning, and business, among other domains, to analyze and discover useful patterns to improve the decision-making process. In general, these tools help to (1) rationalize decisions and the decision-making process, (2) minimize errors and deal with a range of scenarios, (3) get a better understanding of the situation at hand (due to the availability of historical data), (4) assess alternative solutions (based on key performance indicators), and (5) map them against the best possible outcome (benchmarking), which helps make decision-making more agile. For this reason, data analysis has been called "the sexiest job of the twenty-first century." For example, big data jobs in the United States are estimated to be 500,000. But the McKinsey Global Institute (2016) estimates that there is still a shortage of between 140,000 and 190,000 workers with a background in statistics, computer engineering, and other applied fields, and that 1.5 million managers are needed to evaluate and make decisions on big data. This means that 50–60% of the required staff was lacking in the year 2018 in the United States alone (Dean 2018).

Some areas and cases where statistical and machine learning techniques have been successfully applied to create knowledge and make sense of data are given next. For example, in astronomy these techniques have been used for the classification of exoplanets using thousands of images taken of these celestial bodies. Banks use them to decide if a client will be granted credit or not, using as predictors many socioeconomic variables they ask of clients. Banks also use them to detect credit card fraud. On the internet, they are used to classify emails as spam or ham (not spam) based on previous emails and the text they contain. In genomic selection, they are used to predict grain yield (or another trait) of non-phenotyped plants using

information, including thousands of markers and environmental data. In Google, they are used for recommending books, movies, products, etc., using previous data on the characteristics (age, gender, location, etc.) of the people who have used these services. They are also used in self-driving cars, that is, cars capable of sensing their environment and moving with little or no human input, based on thousands of images and information from sensors that perceive their surroundings. These examples give more evidence that the appropriate use of data is a powerful weapon for getting knowledge of the target population.

However, as is true of any new technology, this data-related technology can be used against society. One example is the Facebook–Cambridge Analytica data scandal that was a major political scandal in early 2018 when it was revealed that Cambridge Analytica had harvested personal data from the Facebook profiles of millions of people without their consent and used them for political purposes. This scandal was described as a watershed moment in the public understanding of personal data and precipitated a massive fall in Facebook's stock price and calls for tighter regulation of tech companies' use of data. For these reasons, some experts believe that governments have a responsibility to create and enforce rules on data privacy, since data are a powerful weapon, and weapons should be controlled, and because privacy is a fundamental human right. All these are very important to avoid the weaponization of data against people and society.

To take advantage of the massive data collected in Genomic Selection (GS) and many other domains, it is really important to train people in statistical machine learning methods and related areas to perform precise prediction, extract useful knowledge, and find hidden data patterns. This means that experts in statistical machine learning methods should be able to identify the statistical or machine learning method that is most relevant to a given problem, since there is no universal method that works well for all data sets, cleans the original data, implements these methods in statistical machine learning software, and interprets the output of statistical machine learning methods correctly to translate the big data collected into insights and operational value quickly and accurately.

1.2 Genomic Selection

Plant breeding is a key scientific area for increasing the food production required to feed the people of our planet. The key step in plant breeding is selection, and conventional breeding is based on phenotypic selection. Breeders choose good offspring using their experience and the observed phenotypes of crops, so as to achieve genetic improvement of target traits (Wang et al. 2018). Thanks to this area (and related areas of science), the genetic gain nowadays has reached a near-linear increase of 1% in grain yield yearly (Oury et al. 2012; Fischer et al. 2014). However, a linear increase of at least 2% is needed to cope with the 2% yearly increase in the world population, which relies heavily on wheat products as a source of food (FAO 2011). For this reason, genomic selection (GS) is now being implemented in many

plant breeding programs around the world. GS consists of genotyping (markers) and phenotyping individuals in the reference (training) population and, with the help of statistical machine learning models, predicting the phenotypes or breeding values of the candidates for selection in the testing (evaluation) population that were only genotyped. GS is revolutionizing plant breeding because it is not limited to traits determined by a few major genes and allows using a statistical machine learning model to establish the associations between markers and phenotypes and also to make predictions of non-phenotyped individuals that help make a more comprehensive and reliable selection of candidate individuals. In this way, it is essential for accelerating genetic progress in crop breeding (Montesinos-López et al. 2019).

1.2.1 Concepts of Genomic Selection

The development of different molecular marker systems that started in the 1980s drastically increased the total number of polymorphic markers available to breeders and molecular biologists in general. The single nucleotide polymorphism (SNP) that has been intensively used in QTL discovery is perhaps the most popular high-throughput genotyping system (Crossa et al. 2017). Initially, by applying marker-assisted selection (MAS), molecular markers were integrated with traditional phenotypic selection. In the context of simple traits, MAS consists of selecting individuals with QTL-associated markers with major effects; markers not significantly associated with a trait are not used (Crossa et al. 2017). However, after many attempts to improve complex quantitative traits by using QTL-associated markers, there is not enough evidence that this method really can be helpful in practical breeding programs due to the difficulty of finding the same QTL across multiple environments (due to QTL $\times$ environment interaction) or in different genetic backgrounds (Bernardo 2016). Due to this difficulty of the MAS approach, in the early 2000s, an approach called association mapping appeared with the purpose of overcoming the insufficient power of linkage analysis, thus facilitating the detection of marker–trait associations in non-biparental populations and fine-mapping chromosome segments with high recombination rates (Crossa et al. 2017). However, even the fine-mapping approach was unable to increase the power to detect rare variants that may be associated with economically important traits.

For this reason, Meuwissen et al. (2001) proposed the GS methodology (that was initially used in animal science), which is different from association mapping and QTL analysis, since GS simultaneously uses all the molecular markers available in a training data set for building a prediction model; then, with the output of the trained model, predictions are performed for new candidate individuals not included in the training data set, but only if genotypic information is available for those candidate individuals. This means that the goal of GS is to predict breeding and/or genetic values. Because GS is implemented in a two-stage process, to successfully implement it, the data must be divided into a training (TRN) and a testing (TST) set, as can be observed in Fig. 1.1. The training set is used in the first stage, while the testing set

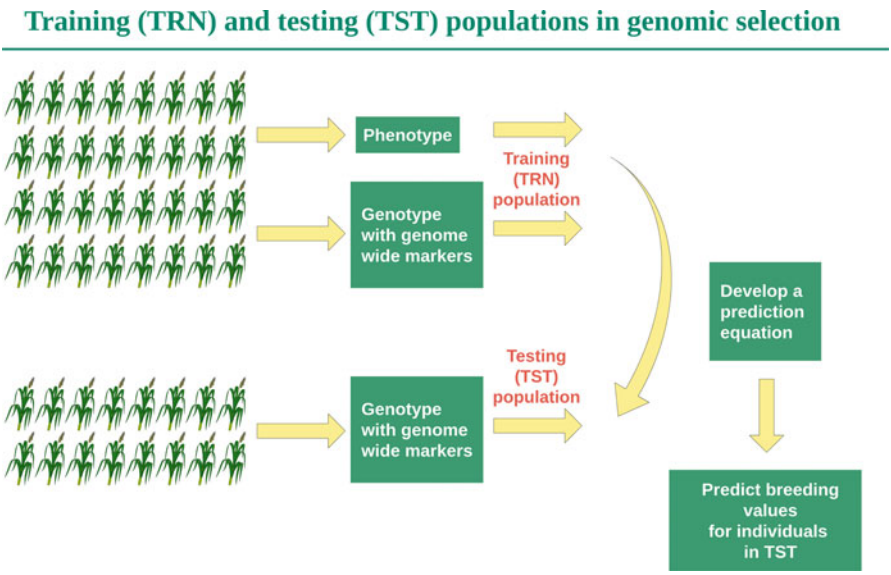

Fig. 1.1 Schematic representation of TRN and TST sets required for implementing GS (Crossa et al. 2017)

is used in the second stage. The main characteristics of the training set are (a) it combines molecular (independent variables) and phenotypic (dependent variables) data and (b) it contains enough observations (lines) and predictors (molecular data) to be able to train a statistical machine learning model with high generalized power (able to predict data not used in the training process) to predict new lines. The main characteristic of the testing set is that it only contains genotypic data (markers) for a sample of observations (lines) and the goal is to predict the phenotypic or breeding values of lines that have been genotyped but not phenotyped.

The two basic populations in a GS program are shown in Fig. 1.1: the training (TRN) data whose genotype and phenotype are known and the testing (TST) data whose phenotypic values are to be predicted using their genotypic information. GS substitutes phenotyping for a few selection cycles. Some advantages of GS over traditional (phenotypic) selection are that it: (a) reduces costs, in part by saving the resources required for extensive phenotyping, (b) saves time needed for variety development by reducing the cycle length, (c) has the ability to substantially increase the selection intensity, thus providing scenarios for capturing greater gain per unit time, (d) makes it possible to select traits that are very difficult to measure, and (e) can improve the accuracy of the selection process. Of course, successful implementation of GS depends strongly on the quality of the training and testing sets.

GS has great potential for quickly improving complex traits with low heritability, as well as significantly reducing the cost of line and hybrid development. Certainly, GS could also be employed for simple traits with higher heritability than complex traits, and high genomic prediction (GP) accuracy is expected. Application of GS in

plant breeding could be limited by some of the following factors: (i) genotyping cost, (ii) unclear guidelines about where in the breeding program GS could be efficiently applied (Crossa et al. 2017), (iii) insufficient number of lines or animals in the reference (training) population, (iv) insufficient number of SNPs in the panels, and (v) the reference population contains very heterogeneous individuals (plants or animals).

1.2.2 Why Is Statistical Machine Learning a Key Element of Genomic Selection?

GS is challenging and very interesting because it aims to improve crop productivity to satisfy humankind's need for food. Addressing the current challenges to increase crop productivity by improving the genetic makeup of plants and avoiding plant diseases is not new, but it is of paramount importance today to be able to increase crop productivity around the world without the need to increase the arable land. Statistical machine learning methods can help improve GS methodology, since they are able to make computers learn patterns that could be used for analysis, interpretation, prediction, and decision-making. These methods learn the relationships between the predictors and the target output using statistical and mathematical models that are implemented using computational tools to be able to predict (or explain) one or more dependent variables based on one or more independent variables in an efficient manner. However, to do this successfully, many real-world problems are only approximated using the statistical machine learning tools, by evaluating probabilistic distributions, and the decisions made using these models are supported by indicators like confidence intervals. However, the creation of models using probability distributions and indicators for evaluating prediction (or association) performance is a field of statistical machine learning, which is a branch of artificial intelligence, understanding as statistical machine learning the application of statistical methods to identify patterns in data using computers, but giving computers the ability to learn without being explicitly programmed (Samuel 1959). However, artificial intelligence is the field of science that creates machines or devices that can mimic intelligent behaviors.

As mentioned above, statistical machine learning allows learning the relationship between two types of information that are assumed to be related. Then one part of the information (input or independent variables) can be used to predict the information lacking (output or dependent variables) in the other using the learned relationship. The information we want to predict is defined as the response variable (y), while the information we use as input are the predictor variables ($\boldsymbol{X}$). Thanks to the continuous reduction in the cost of genotyping, GS nowadays is implemented in many crops around the world, which has caused the accumulation of large amounts of biological data that can be used for prediction of non-phenotyped plants and animals. However, GS implementation is still challenging, since the quality of the data (phenotypic and

genotypic) needs to be improved. Many times the genotypic information available is not enough to make high-quality predictions of the target trait, since the information available has a lot of noise. Also, since there is no universal best prediction model that can be used under all circumstances, a good understanding of statistical machine learning models is required to increase the efficiency of the selection process of the best candidate individuals with GS early in time. This is very important because one of the key components of genomic selection is the use of statistical machine learning models for the prediction of non-phenotyped individuals. For this reason, statistical machine learning tools have the power to help increase the potential of GS if more powerful statistical machine learning methods are developed, if the existing methods can deal with larger data sets, and if these methods can be automatized to perform the prediction process with only a limited knowledge of the subject.

For these reasons, statistical machine learning tools promise considerable benefits for GS and agriculture through their contribution to productivity growth and gain in the genetic makeup of plants and animals without the need to increase the arable land. At the same time, with the help of statistical machine learning tools, GS is deeply impacting the traditional way of selecting candidate individuals in plant and animal breeding. Since GS can reduce by at least half the time needed to select candidate individuals, it has been implemented in many crops around the globe and is radically changing the traditional way of developing new varieties and animals all over the world.

Although GS is not the dominant paradigm for developing new plants and animals, it has the potential to transform the way they are developed due to the following facts: (a) the massive amounts of data being generated in plant breeding programs are now available to train the statistical machine learning methods, (b) new technologies such as sensors, satellite technology, and robotics allow scientists to generate not only genomic data but also phenotypic data that can capture a lot of environmental and phenotypic information that can be used in the modeling process to increase the performance of statistical machine learning methods, (c) increased computational power now allows complex statistical machine learning models with larger data sets to be implemented in less time, and (d) there is now greater availability of user-friendly statistical machine learning software for implementing a great variety of statistical machine learning models.

However, there are still limitations for the successful implementation of GS with the help of statistical machine learning methods because much human effort is required to collect a good training data set for supervised learning. Although nowadays it is possible to measure a lot of independent variables (markers, environmental variables) due to the fact that the training set should be measured in real-world experiments conducted in different environments and locations, this is expensive and subject to nature's random variability. This means that GS data are hampered by issues such as multicollinearity among markers (adjacent markers are highly correlated) and by a problem that consists of having a small number of observations and a large number of independent variables (commonly known as "large p small n"), which poses a statistical challenge. For this reason, obtaining data sets that are large and comprehensive enough to be used for training—for example,

creating or obtaining sufficient plant trial data to predict yield, plant height, grain quality, and presence or absence of disease outcomes more accurately—is also often challenging.

Another challenge is that of building statistical machine learning techniques that are able to generalize the unseen data, since statistical machine learning methods continue to have difficulty carrying their experiences from one set of circumstances to another. This is known as transfer learning, and it focuses on storing knowledge gained when training a particular machine learning algorithm and then using this stored knowledge for solving another related problem. In other words, transfer learning is still very challenging and occurs when a statistical machine learning model is trained to accomplish a certain task and then quickly apply that learning exercise to a different activity.

Another disadvantage is that even though today there are many software programs for implementing statistical machine learning tools for GS, the computational resources required for learning from moderate to large data sets are very expensive and most of the time it is not possible to implement them in commonly used computers, since servers with many cores and considerable computational resources are required. However, the rapid increase in computational power will change this situation in the coming years.

1.3 Modeling Basics

1.3.1 What Is a Statistical Machine Learning Model?

A model is a simplified description, using mathematical tools, of the processes we think that give rise to the observations in a set of data. A model is deterministic if it explains (completely) the dependent variables based on the independent ones. In many real-world scenarios, this is not possible. Instead, statistical (or stochastic) models try to approximate exact solutions by evaluating probabilistic distributions. For this reason, a statistical model is expressed by an equation composed of a *systematic* (deterministic) and a *random part* (Stroup 2012) as given in the next equation:

$$y_i = f(\boldsymbol{x_i}) + \epsilon_i, \text{for } i = 1, 2, \ldots, n, \tag{1.1}$$

where y_i represents the response variable in individual i and $f(\boldsymbol{x}_i)$ is the systematic part of the model because it is *determined* by the explanatory variables (predictors). For these reasons, the systematic part of the statistical learning model is also called the deterministic part of the model, which gives rise to an unknown mathematical function (f) of $\boldsymbol{x}_i = x_{i1}, \ldots, x_{ip}$ not subject to random variability. ϵ_i is the ith random element (error term) which is independent of $\boldsymbol{x_i}$ and has mean zero. The ϵ_i term tells us that observations are assumed to vary at random about their mean, and it also defines the uniqueness of each individual. In theory (at least in some philosophical

domains), if we know the mechanism that gives rise to the uniqueness of each individual, we can write a completely deterministic model. However, this is rarely possible because we use probability distributions to characterize the observations measured in the individuals. Most of the time, the error term (ϵ_i) is assumed to follow a normal distribution with mean zero and variance σ^2 (Stroup 2012).

As given in Eq. (1.1), the f function that gives rise to the systematic part of a statistical learning model is not restricted to a unique input variable, but can be a function of many, or even thousands, of input variables. In general, the set of approaches for estimating f is called statistical learning (James et al. 2013). Also, the functions that f can take are very broad due to the huge variety of phenomena we want to predict and due to the fact that there is no universally superior f that can be used for all processes. For this reason, to be able to perform good predictions out of sample data, many times we need to fit many models and then choose the one most likely to succeed with the help of cross-validation techniques. However, due to the fact that models are only a simplified picture of the true complex process that gives rise to the data at hand, many times it is very hard to find a good candidate model. For this reason, statistical machine learning provides a catalog of different models and algorithms from which we try to find the one that best fits our data, since there is no universally best model and because there is evidence that a set of assumptions that works well in one domain may work poorly in another—this is called the *no free lunch theorem* by Wolpert (1996). All these are in agreement with the famous aphorism, "all models are wrong, but some are useful," attributed to the British statistician George Box (October 18, 1919–March 28, 2013) who first mentioned this aphorism in his paper "Science and Statistics" published in the *Journal of the American Statistical Association* (Box 1976). As a result of the *no free lunch theorem*, we need to evaluate many models, algorithms, and sets of hyperparameters to find the best model in terms of prediction performance, speed of implementation, and degree of complexity. This book is concerned precisely with the appropriate combination of data, models, and algorithms needed to reach the best possible prediction performance.

1.3.2 The Two Cultures of Model Building: Prediction Versus Inference

The term "two cultures" in statistical model building was coined by Breiman (2001) to explain the difference between the two goals for estimating f in Eq. (1.1): prediction and inference. These definitions are provided in order to clarify the distinct scientific goals that follow inference and empirical predictions, respectively. A clear understanding and distinction between these two approaches is essential for the progress of scientific knowledge. Inference and predictive modeling reflect the process of using data and statistical (or data mining) methods for inferring or predicting, respectively. The term modeling is intentionally chosen over model to

highlight the entire process involved, from goal definition, study design, and data collection to scientific use (Breiman 2001).

Prediction

The prediction approach can be defined as the process of applying a statistical machine learning model or algorithm to data for the purpose of predicting new or future observations. For example, in plant breeding a set of inputs (marker information) and the outcome Y (disease resistance: yes or no) are available for some individuals, but for others only marker information is available. In this case, marker information can be used as a predictor and the disease status should be used as the response variable. When scientists are interested in predicting new plants not used to train the model, they simply want an accurate model to predict the response using the predictors. However, when scientists are interested in understanding the relationship between each individual predictor (marker) and the response variable, what they really want is a model for inference. Another example is when forest scientists are interested in developing models to predict the number of fire hotspots from an accumulated fuel dryness index, by vegetation type and region. In this context, it is obvious that scientists are interested in future predictions to improve decision-making in forest fire management. Another example is when an agro-industrial engineer is interested in developing an automated system for classifying mango species based on hundreds of mango images taken with digital cameras, mobile phones, etc. Here again it is clear that the best approach to build this system should be based on prediction modeling since the objective is the prediction of new mango species, not any of those used for training the model.

Inference

Many areas of science are devoted mainly to testing causal theories. Under this framework, scientists are interested in testing the validity of a theoretical causal relationship between the causal variables (X; underlying factors) and the measured variable (Y) using statistical machine learning models and collected data to test the causal hypotheses. The type of statistical machine learning models used for testing causal hypotheses are usually association-based models applied to observational data (Shmueli 2012). For example, regression models are one type of association-based models used for testing causal hypotheses. This practice is justified by the theory itself, which assumes the causality. In this context, the role of the theory is very strong and the reliance on data and statistical modeling is strictly through the lens of the theoretical model. The theory–data relationship varies in different fields. While the social sciences are very theory-heavy, in areas such as bioinformatics and natural language processing, the emphasis on a causal theory is much weaker. Hence, given this reality, Shmueli (2012) defined *explaining* as causal explanation and *explanatory modeling* as the use of statistical models for testing causal explanations.

Next, we provide some great examples used for testing causal hypotheses: for example, between 1911 and 1912, Austrian physicist Victor Hess made a series of ten balloon ascents to study why metal plates tend to charge spontaneously. At that

time, it was assumed that the cause was the presence, in small quantities, of radioactive materials in rocks. If this is the case, as one moves away from the ground, the tendency of metal plates to be charged should decrease. Hess brought with him three electroscopes, which are instruments composed basically of two metal plates enclosed in a glass sphere. When charging the plates, they separated one from the other. Hess observed that from a certain height, the three electroscopes tended to be charged to a greater extent. On August 7, 1912, together with a flight commander and a meteorologist, he made a 6-hour flight in which he ascended to more than 5000 m in height (Schuster 2014). Hess published his results in 1913, where he presented his conclusion that the cause of the charge of the electroscopes was radiation of cosmic origin that penetrates the atmosphere from above. The discovery of this cosmic radiation, for which Hess received the Nobel Prize in 1936, opened a new window for the study of the universe (Schuster 2014). It was in 1925 when American physicist Robert Andrew Millikan introduced the term "cosmic rays" to describe this radiation, and what it was made of was still unknown. In this example, it is clear that the goal of the analysis was association.

No one suspected that tobacco was a cause of lung tumors until the final decade of the nineteenth century. In 1898, Hermann Rottmann (a medical student) in Würzburg proposed that tobacco dust—not smoke—might be causing the elevated incidence of lung tumors among German tobacco workers. This was a mistake corrected by Adler (1912) who proposed that smoking might be to blame for the growing incidence of pulmonary tumors. Lung cancer was still a very rare disease; so rare, in fact, that medical professors, when confronted with a case, sometimes told their students they might never see another. However, in the 1920s, surgeons were already faced with a greater incidence of lung cancer, and they began to get confused about its possible causes. In general, smoking was blamed, along with asphalt dust from recently paved roads, industrial air pollution, and the latent effects of poisonous gas exposure during World War I or the global influenza pandemic of 1918–1919. These and many other theories were presented as possible explanations for the increase in lung cancer, until evidence from multiple research sources made it clear that tobacco was the main culprit (Proctor 2012). Here again it is clear that the goal of the analysis should be related to inference.

1.3.3 Types of Statistical Machine Learning Models and Model Effects

1.3.3.1 Types of Statistical Machine Learning Models

Statistical machine learning models are most commonly classified as parametric models, semiparametric models, and nonparametric models. Next, we define each type of statistical machine learning models and provide examples that help to understand each one.

Parametric Model It is a type of statistical machine learning model in which all the predictors take predetermined forms with the response. Linear models (e.g., multiple regression: $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$), generalized linear models [Poisson regression: $E(y|x) = \exp(\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)$], and nonlinear models (nonlinear regression: $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 e^{\beta_4 x_3} + \epsilon$) are examples of parametric statistical machine learning models because we know the function that describes the relationship between the response and the explanatory variables. These models are very easy to interpret but very inflexible.

Nonparametric Model It is a type of statistical machine learning model in which none of the predictors take predetermined forms with the response but are constructed according to information derived from data. Two common statistical machine learning models are kernel regression and smoothing spline. Kernel regression estimates the conditional expectation of y at a given value x using a weighted filter on the data ($y = m(x) + \epsilon$, with $\widehat{m}(x_0) = \frac{\sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right) y_i}{\sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right)}$), where h is the bandwidth (this estimator of $m(x)$ is called the *Nadaraya–Watson (NW) kernel estimator*) and K is a kernel function. While smoothing splines minimize the sum of squared residuals plus a term which penalizes the roughness of the fit [$y = \beta_0 + \beta_1 x + \beta_2 x^2 + 3x^3 + \sum_{j=1}^{J} \beta_{1j} \left(x - \theta_j\right)_+^3$, where $(x - \theta_j)_+ = x - \theta_j$, $x > \theta_j$ and 0 otherwise], this model in brackets is a spline of degree 3 which is represented as a power series. These models are very difficult to interpret but are very flexible. Nonparametric statistical machine learning models differ from parametric models in that the shape of the functional relationships between the response (dependent) and the explanatory (independent) variables are not predetermined but can be adjusted to capture unusual or unexpected features of the data. Nonparametric statistical machine learning models can reduce modeling bias (the difference between estimated and true values) by imposing no specific model structure other than certain smoothness assumptions, and therefore they are particularly useful when we have little information or we want to be flexible about the underlying statistical machine learning model. In general, nonparametric statistical machine learning models are very flexible and are better at fitting the data than parametric statistical machine learning models. However, these models require larger samples than parametric statistical machine learning models because the data must supply the model structure as well as the model estimates.

Semiparametric Model It is a statistical machine learning model in which *part* of the predictors do not take predetermined forms while the other part takes known forms with the response. Some examples are (a) $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + m(x) + \epsilon$ and (b) $y = \exp(\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) + m(x) + \epsilon$. This means that semiparametric models are a mixture of parametric and nonparametric models.

When the relationship between the response and explanatory variables is known, parametric statistical machine learning models should be used. If the relationship is

unknown and nonlinear, nonparametric statistical machine learning models should be used. When we know the relationship between the response and part of the explanatory variables, but do not know the relationship between the response and the other part of the explanatory variables, we should use semiparametric statistical machine learning models. Any application area that uses statistical machine learning analysis could potentially benefit from semi/nonparametric regression.

1.3.3.2 Model Effects

Many statistical machine learning models are expressed as models that incorporate *fixed effects*, which are parameters associated with an entire population or with certain levels of experimental factors of interest. Other models are expressed as *random effects*, where individual experimental units are drawn at random from a population, while a model with *fixed effects* and *random effects* is called a *mixed-effects* model (Pinheiro and Bates 2000).

According to Milliken and Johnson (2009), a factor is a *random effect* if its levels consist of a random sample of levels from a population of possible levels, while a factor is a *fixed effect* if its levels are selected by a nonrandom process or if its levels consist of the entire population of possible levels.

Mixed-effects models, also called multilevel models in the social science community (education, psychology, etc.), are an extension of regression models that allow for the incorporation of random effects; they are better suited to describe relationships between a response variable and some covariates in data that are grouped according to one or more classification factors. Examples of such grouped data include longitudinal data, repeated measures data, multilevel data, and block designs. One example of grouped data are animals that belong to the same herd; for example, assume we have 10 herds with 50 animals (observations) in each. By associating to observations (animals) sharing the same level of a classification factor (herd) a common random effect, mixed-effects models parsimoniously represent the covariance structure induced by the grouping of data (Pinheiro and Bates 2000). Most of the early work on mixed models was motivated by the animal science community driven by the need to incorporate heritabilities and genetic correlations in parsimonious fashion.

Next we provide an example to illustrate how to build these types of models. Assume that five environments were chosen at random from an agroecological area of Mexico. Then in each area, three replicates of a new variety (NV) of maize were tested to measure grain yield (GY) in tons per hectare. The data collected from this experiment are shown in Fig. 1.2.

Since the only factor that changes among the observations measured in this experiment is the environment, they are arranged in a one-way classification because they are classified according to a single characteristic: the environments in which the observations were made (Pinheiro and Bates 2000). The data structure is very simple since each row represents one observation for which the environment and GY were recorded, as can be seen in Table 1.1.

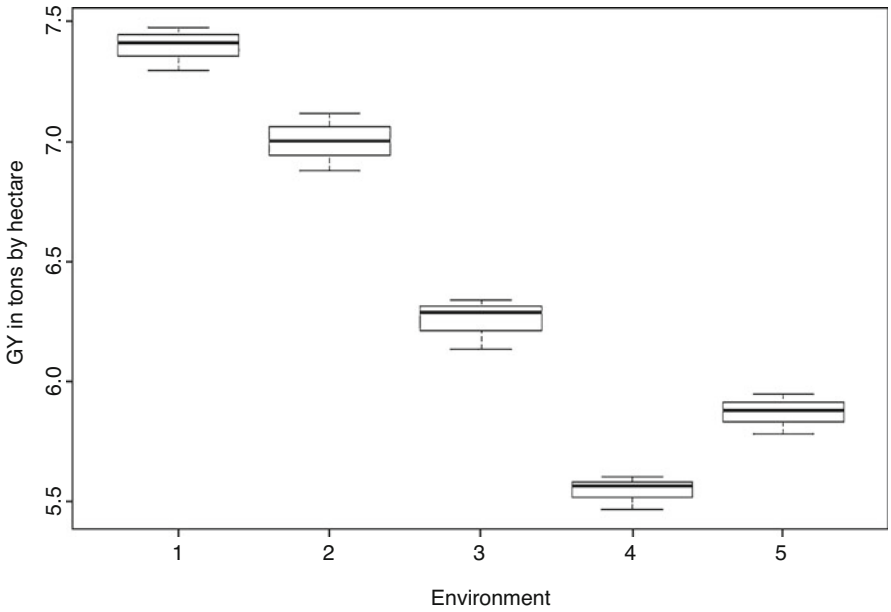

Fig. 1.2 Grain yield (GY) in tons per hectare by environment

Table 1.1 Grain yield (GY) measured in five environments (Env) with three repetitions (Rep) in each environment

Env	Rep	GY
1	1	7.476
1	2	7.298
1	3	7.414
2	1	7.117
2	2	6.878
2	3	7.004
3	1	6.136
3	2	6.340
3	3	6.288
4	1	5.600
4	2	5.564
4	3	5.466
5	1	5.780
5	2	5.948
5	3	5.881

The breeder who conducted this experiment was only interested in the average GY for a typical environment, that is, the expected GY, the variation in average GY among environments (between-environment variability), and the variation in the observed GY for a single environment (within-environment variability). Figure 1.2 shows that there is considerable variability in the mean GY for different

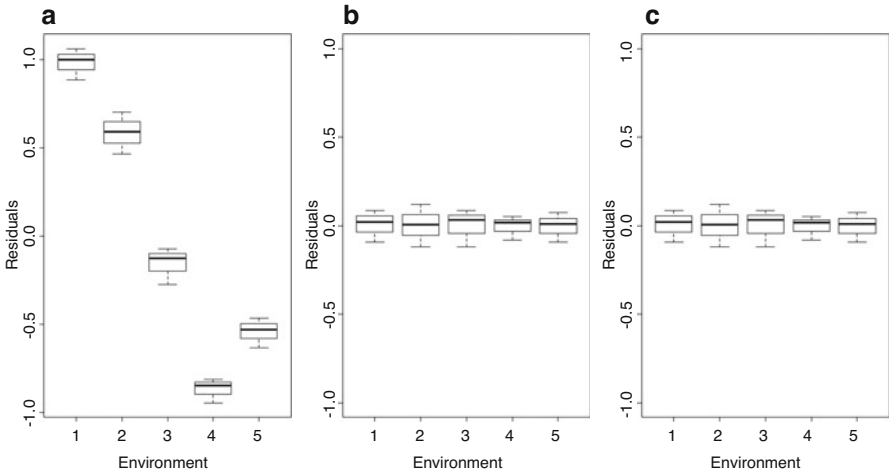

Fig. 1.3 Box plot of residuals by environments of the fit (**a**) of a single-mean model, (**b**) of a fixed-effects model with environment effect, and (**c**) of a mixed-effects model with environment effect

environments and that the within-environment variability is smaller than the between-environment variability.

Data from a one-way classification like the one given in Table 1.1 can be analyzed under both approaches with fixed effects or random effects. The decision on which approach to use depends basically on the goal of the study, since if the goal is to make inferences about the population for which these environments (levels) were drawn, then the random effect approach is the best option, but if the goal is to make inferences about the particular environments (levels) selected in this experiment, then the fixed-effects approach should be preferred.

Assuming a simple model that ignores the environment

$$GY_{ij} = \beta + e_{ij}, \qquad i = 1, .., 5, j = 1, 2, 3, \tag{1.2}$$

where GY_{ij} denotes the GY of environment i in replication j, and β is the mean GY across the population of environments sampled with $e_{ij} \sim N(0, \sigma^2)$. Using this model we estimate $\widehat{\beta} = 6.4127$ and the residual standard error is equal to $\widehat{\sigma} = 0.7197$. By fitting this model and observing the boxplot of the residuals (Fig. 1.3a) versus the environments, we can see that these residuals are very large and different for each environment, which can be attributed to the fact that this model ignores the environmental effect and was only fitted as a single-mean model, which implies that the environmental effects are included in the residuals. For this reason, we then incorporated the environmental effect in the model as a separate effect. This fixed-effects model is equal to a one-way classification model as

$$GY_{ij} = \beta_i + e_{ij}, \quad i = 1, .., 5, j = 1, 2, 3, \tag{1.3}$$

where β_i represents the fixed effect of environment i, while the other terms in the model are the same as those in Eq. (1.2). By fitting this model using least squares, we now have a beta coefficient for each environment equal to: $\widehat{\beta}_1 = 7.396$, $\widehat{\beta}_2 = 6.999$, $\widehat{\beta}_3 = 6.255$, $\widehat{\beta}_4 = 5.543$, $\widehat{\beta}_5 = 5.869$, while the residual standard error is equal to $\widehat{\sigma} = 0.095$. Figure 1.3b shows that the residuals are considerably lower and more centered around zero than in the first fitted model (single-mean model), which can be observed in the residual standard error that is 7.47 times smaller. Therefore, we have evidence that the model given in Eq. (1.3) successfully accounted for the environmental effects. Two drawbacks of the model with fixed effects given in Eq. (1.3) are that it is unable to provide an estimate of the between-environments variability and that the number of parameters in the model increases linearly with the number of environments. Fortunately, the random effects model circumvents these problems by treating the environmental effects as random variations around a population mean. Next we reparameterize model (1.3) as a random effects model. We write

$$GY_{ij} = \overline{\beta} + \left(\beta_i - \overline{\beta}\right) + e_{ij}, \tag{1.4}$$

where $\overline{\beta} = \sum_{i=1}^{5} \beta_i / 5$ represents the average grain yield for the environments in the experiment. The random effects model replaces $\overline{\beta}$ with the mean grain yield across the population of environments and replaces the deviations $\beta_i - \overline{\beta}$ with the random variables whose distribution is to be estimated. If $\beta_i - \overline{\beta}$ is not assumed random the model belongs to fixed effects. Therefore, the random effects version of the model given in Eq. (1.4) is equal to

$$GY_{ij} = \beta + b_i + e_{ij}, \tag{1.5}$$

where β is the mean grain yield across the population of environments, b_i is a random variable representing the deviation from the population mean of the grain yield for the ith environment, and e_{ij} is defined as before. To complete this statistical machine learning model, we must specify the distribution of the random variables b_i, with $i = 1, \ldots, 5$. It is common to assume that b_i is normally distributed with mean zero and variance between environments σ_b^2, that is, b_i is distributed $N\left(0, \sigma_b^2\right)$. It is also common to assume independence between the two random effects b_i and e_{ij}. Models with at least two sources of random variation are also called hierarchical models or multilevel models (Pinheiro and Bates 2000). The covariance between observations in the same environment is σ_b^2, which corresponds to a correlation of $\sigma_b^2 / \left(\sigma_b^2 + \sigma^2\right)$. The parameters of the mixed model given in Eq. (1.5) are β, σ_b^2, and σ^2, and irrespective of the number of environments in the experiment, the required number of parameters will always be three, although the random effects, b_i, behave like parameters. We will, however, require $\widehat{b}_i$ predictions of these random effects, given the observed data at hand. Note that when fitting this model (Eq. 1.5) for the environmental data, the parameter estimates were $\beta = 6.413$, $\sigma_b^2 = 0.594$, and $\widehat{\sigma} = 0.095$. It is evident that there was no improvement in terms of fitting since the plot of

residuals versus environment looks the same as the last fitted model (Fig. 1.3c) and the estimated residual standard error was the same as that under the fixed-effects model, which includes the effects of environment in the predictor, but as mentioned above, many times requires considerably fewer parameter estimates than a fixed-effects model.

1.4 Matrix Algebra Review

In this section, we provide the basic elements of linear algebra that are key to understanding the machinery behind the process of building statistical machine learning algorithms.

A ***matrix*** is a rectangular arrangement of numbers whose elements can be identified by the row and column in which they are located. For example, matrix $\boldsymbol{E}$, consisting of three rows and five columns, can be represented as follows:

$$\boldsymbol{E} = \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{14} & E_{15} \\ E_{21} & E_{22} & E_{23} & E_{24} & E_{25} \\ E_{31} & E_{32} & E_{33} & E_{34} & E_{35} \end{bmatrix}$$

For example, by replacing the matrix with numbers, we have

$$\boldsymbol{E} = \begin{bmatrix} 7 & 9 & 4 & 3 & 6 \\ 9 & 5 & 9 & 8 & 11 \\ 3 & 2 & 11 & 9 & 6 \end{bmatrix}$$

where the element E_{ij} is called the *ij*th element of the matrix; the first subscript refers to the row where the element is located and the second subscript refers to the column, for example, $E_{32} = 2$. The order of an array is the number of rows and columns. Therefore, a matrix with r rows and c columns has an order of $r \times c$. Matrix $\boldsymbol{E}$ has an order of 3×5 and is denoted as $\boldsymbol{E}_{3 \times 5}$.

In R, the way to establish an array is through the command *matrix(...)* with parameters of this function given by *matrix* (*data* = *NA, nrow* = *3, ncol* = *5, byrow* = *FALSE*) where *data* is the data for the matrix, *nrow* the number of rows, *ncol* the number of columns, and *byrow* is the way in which you will accommodate the data in the matrix by row or column. The data entered by default are FALSE, so they will fill the matrix by columns, while if you specified TRUE, they will fill the matrix by rows.

For example, to build matrix $\boldsymbol{E}$ in R, use the following R script:

$$\boldsymbol{E} = \begin{bmatrix} 7 & 9 & 4 & 3 & 6 \\ 9 & 5 & 9 & 8 & 11 \\ 3 & 2 & 11 & 9 & 6 \end{bmatrix}$$

```
E <- matrix(data= c(7,9,4,3,6,9,5,9,8,11,3,2,11,9,6), nrow = 3, ncol = 5, byrow = TRUE)
E
     [,1] [,2] [,3] [,4] [,5]
[1,]   7   9   4   3   6
[2,]   9   5   9   8  11
[3,]   3   2  11   9   6
```

To access all the values of a row, for example, the first row of matrix $\boldsymbol{E}$, you can use:

```
E[1,]
[1] 7 9 4 3 6
```

While, to access all the values of a column, for example, the first column of matrix $\boldsymbol{E}$, you can use:

```
E[,1]
[1] 7 9 3
```

To access a specific element, for example, row 3, column 2 of matrix $\boldsymbol{E}$, you can specify:

```
E[3,2]
[1] 2
```

A matrix consisting of a single row is called a vector. For example, a vector that has five elements can be represented as

$$\boldsymbol{e} = [8 \;\; 10 \;\; 5 \;\; 4 \;\; 7]$$

Here only a subscript is needed to specify the position of an element within the vector. Therefore, the i*th* element in vector $\boldsymbol{e}$ refers to the element in the ith column. For example, $e_3 = 5$.

To create a vector in R, we use the c (...) command that receives the data, separated by a comma. For example, the vector named $\boldsymbol{e}$ can be created using the following command:

```
e <- c(8,10,5,4,7)
e
[1] 8 10 5 4 7
```

To access a specific index, you specify the value between brackets, for example, index 3 of vector ***e***.

```
e[3]
```

```
[1] 5
```

Next we provide definitions and examples of some common types of matrices. We start with a ***square matrix***, which is a matrix with the same number of rows and columns. A matrix ***B*** of order 3×3 is shown below.

$$\boldsymbol{B} = \begin{bmatrix} 3 & 5 & 0 \\ 5 & 1 & 5 \\ -2 & -3 & 7 \end{bmatrix}$$

The *ij* elements in the square matrix where *i* equals *j* are called diagonal elements. The rest of the elements are known as elements that are outside the diagonal, so in this example, the elements of the diagonal of matrix ***B*** are 3, 1, and 7.

To create this type of matrix in R, simply use the *matrix* (...) command and specify the dimensions in the *ncol* and *nrow* parameters, as in the following command:

```
B <- matrix(data = c(3,5,-2,5,1,-3,0,5,7), nrow = 3, ncol = 3)
B
```

```
     [,1] [,2] [,3]
[1,]    3    5    0
[2,]    5    1    5
[3,]   -2   -3    7
```

Next we define a ***diagonal matrix*** as a square matrix that has zeros in all the elements that are outside the diagonal. For example, by extracting the diagonal elements of the above matrix (***B***), we can form the following diagonal matrix:

$$\boldsymbol{D} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

Another special matrix is when the elements of the diagonal are 1; it is called an ***identity matrix***, and is usually denoted as $\boldsymbol{I}_r$, and *r* denotes the order of the matrix.

In R there is the command *diag (x = 1, ncol, nrow)* that works to create a diagonal matrix, but if you want to extract the diagonal of the ***B*** matrix, it can be extracted in the following way:

```
diag(B)
[1] 3 1 7
```

If we want to create an identity matrix, it is enough to specify the size of the matrix. For example, to create an identity matrix of order 5×5, the command to use would be the following:

```
I <- diag(5)
I
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

A square matrix with all the elements above the diagonal equal to 0 is known as a *lower triangular* matrix; when all the lower elements of the diagonal are 0, it is known as an *upper triangular* matrix. In the example given below, matrix F illustrates a lower triangular matrix and matrix G illustrates an upper triangular matrix.

$$F = \begin{bmatrix} 8 & 0 & 0 \\ 10 & 7 & 0 \\ 4 & 3 & 12 \end{bmatrix}; \qquad G = \begin{bmatrix} 8 & 10 & 5 \\ 0 & 6 & 10 \\ 0 & 0 & 11 \end{bmatrix}$$

The lower triangular matrix can be extracted using the following commands:

```
F <- matrix(data = c(8,10,4,0,7,3,0,0,12), nrow = 3, ncol = 3)
F=F[upper.tri(F)] <- 0
F
      [,1] [,2] [,3]
[1,]    8    0    0
[2,]   10    7    0
[3,]    4    3   12
```

The upper triangular matrix can be extracted using the following commands:

```
G <- matrix(data = c(8,10,5,0,6,10,0,0,11), nrow = 3, ncol = 3, ,byrow=T)
G[lower.tri(G)] <- 0
G
     [,1] [,2] [,3]
[1,]    8   10    5
[2,]    0    6   10
[3,]    0    0   11
```

Next we will illustrate some basic matrix operations. We start by illustrating the ***transpose*** of matrix $\boldsymbol{E}$, commonly indicated as $\boldsymbol{E}'$ or $\boldsymbol{E}^T$; in this type of matrix, the elements *ji* are the *ij* elements of the original matrix. That is, $E'_{ji} = E_{ij}$, where the columns of $\boldsymbol{E}'$ are the rows of $\boldsymbol{E}$, and the rows of $\boldsymbol{E}'$ are the columns of $\boldsymbol{E}$. Below we provide matrix $\boldsymbol{H}$ and its transpose $\boldsymbol{H}'$.

$$\boldsymbol{H} = \begin{bmatrix} 4 & 6 \\ 6 & 2 \\ 0 & -1 \end{bmatrix}; \qquad \boldsymbol{H}' = \begin{bmatrix} 4 & 6 & 0 \\ 6 & 2 & -1 \end{bmatrix}$$

To obtain the transpose of a matrix in $\boldsymbol{H}$, the command *t (...)* is used:

```
H <- matrix(data = c(4,6,0,6,2,-1), nrow = 3,   2)
H
     [,1] [,2]
[1,]    4    6
[2,]    6    2
[3,]    0   -1
```

```
t(H)
 [1,]   4   6   0
[2,]   6   2  -1
```

Two matrices can be added or subtracted only if they have the same number of rows and columns. To demonstrate the adding process, we use the following matrices:

$$\boldsymbol{J} = \begin{bmatrix} 15 & 15 \\ 25 & 35 \end{bmatrix}; \quad \boldsymbol{L} = \begin{bmatrix} 55 & 65 \\ 75 & 85 \end{bmatrix}$$

We form matrix $\boldsymbol{M}$ as the sum of matrices $\boldsymbol{J}$ and $\boldsymbol{L}$, so that $\boldsymbol{M}_{ij} = \boldsymbol{J}_{ij} + \boldsymbol{L}_{ij}$, and their sum is the following:

$$\boldsymbol{M} = \begin{bmatrix} 15+55 & 15+65 \\ 25+75 & 35+85 \end{bmatrix} = \begin{bmatrix} 70 & 80 \\ 100 & 120 \end{bmatrix}$$

Matrix $\boldsymbol{N}$ is reached by subtracting matrices $\boldsymbol{J}$ and $\boldsymbol{L}$, so $\boldsymbol{N}_{ij} = \boldsymbol{J}_{ij} - \boldsymbol{L}_{ij}$, and the subtraction is the following:

$$\boldsymbol{N} = \begin{bmatrix} 15-55 & 15-65 \\ 25-75 & 35-85 \end{bmatrix} = \begin{bmatrix} -40 & -50 \\ -50 & -50 \end{bmatrix}$$

To do the addition and subtraction of matrices in R, it is enough to use the addition or subtraction operator and fulfill the requirement that both matrices have the same dimensions. Next we reproduce the two previous addition and subtraction examples using the commands in R:

```
J <- matrix(data= c(15,15,25,35), ncol = 2, byrow = TRUE)
L <- matrix(data= c(55,65,75,85), ncol = 2, byrow = TRUE)

M<- J+L
M
     [,1] [,2]
[1,]   70   80
[2,]  100  120
```

While the subtraction of matrices is:

```
N<- J-L
N
      [,1]  [,2]
[1,]  -40  -50
[2,]  -50  -50
```

Two matrices can be multiplied only if the number of columns in the first matrix equals the number of rows in the second. The resulting matrix will be equal to the number of rows in the first matrix and the number of columns in the second. Since $\boldsymbol{O} = \boldsymbol{PQ}$, then

$$O = \sum_{j=1}^{m} \sum_{i=1}^{n} \sum_{k=1}^{z} P_{ik} Q_{kj},$$

where m is the number of columns in matrix $\boldsymbol{Q}$, n the number of rows in matrix $\boldsymbol{P}$, and z the number of rows in $\boldsymbol{Q}$ and number of columns in $\boldsymbol{P}$. To demonstrate the above, we have

$$\boldsymbol{P} = \begin{bmatrix} 6 & 8 & 10 \\ 5 & 6 & 8 \\ 4 & 6 & 9 \end{bmatrix}; \boldsymbol{Q} = \begin{bmatrix} 3 & 5 \\ 2 & 2 \\ 9 & 8 \end{bmatrix}$$

Then $\boldsymbol{S}$ is obtained as

$$\begin{aligned} S_{11} &= 6 \times 3 + 8 \times 2 + 10 \times 9 = 124 \\ S_{21} &= 5 \times 3 + 6 \times 2 + 8 \times 9 = 99 \\ S_{31} &= 4 \times 3 + 6 \times 2 + 9 \times 9 = 105 \\ S_{12} &= 6 \times 5 + 8 \times 2 + 10 \times 8 = 126 \\ S_{22} &= 5 \times 5 + 6 \times 2 + 8 \times 8 = 101 \\ S_{32} &= 4 \times 5 + 6 \times 2 + 9 \times 8 = 104 \end{aligned}$$

Therefore,

$$\boldsymbol{S} = \begin{bmatrix} 124 & 126 \\ 99 & 101 \\ 105 & 104 \end{bmatrix}$$

Note that $\boldsymbol{S}$ is of order 3×2, where 3 represents the number of rows in $\boldsymbol{P}$ and 2 equals the number of columns in $\boldsymbol{Q}$.

In order to multiply matrices in R, it is necessary to use the operator *%*%* between the two matrices, in addition to meeting the requirements mentioned above.

```
P <- matrix(data=c(6,5,4,8,6,6,10,8,9), ncol=3)
Q <- matrix(data=c(3,2,9,5,2,8), ncol=2)

S <- P%*%Q
S
     [,1]  [,2]
[1,]  124  126
[2,]   99  101
[3,]  105  104
```

The inverse of a matrix usually is denoted as $\boldsymbol{R}^{-1}$, and when it is multiplied by the original matrix, it results in an identity matrix, that is, $\boldsymbol{R}^{-1}\boldsymbol{R} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix. Only square matrices are invertible.

If it is a diagonal matrix, its inverse can be calculated simply in the following way:

$$\boldsymbol{R} = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

Then

$$\boldsymbol{R}^{-1} = \begin{bmatrix} \frac{1}{7} & 0 & 0 \\ 0 & \frac{1}{8} & 0 \\ 0 & 0 & \frac{1}{6} \end{bmatrix}$$

For a square matrix of 2×2, its inverse can be calculated by finding the determinant, which is the difference between the product of the two elements of the diagonals and the product of the two elements outside the diagonal ($R_{11}R_{22} - R_{12}R_{21}$). Then the position of the elements of the diagonals is reversed by multiplying the elements outside the diagonal by -1 and dividing all the elements by the determinant.

$$\boldsymbol{R} = \begin{bmatrix} 4 & 2 \\ 1 & 6 \end{bmatrix}$$

Then

$$\boldsymbol{R}^{-1} = \frac{1}{(4 \times 6) - (1 \times 2)} \begin{bmatrix} 6 & -2 \\ -1 & 4 \end{bmatrix} = \begin{bmatrix} 0.2727 & -0.0909 \\ -0.0455 & 0.1818 \end{bmatrix}$$

To obtain the inverse in $\boldsymbol{R}$, the solve(...) command is the function used to perform this process, as shown in the following example, where the inverse of the matrix $\boldsymbol{R}$ is obtained.

```
R <- matrix(data = c(4, 1, 2, 6), ncol = 2)
R
     [,1] [,2]
[1,]    4    2
[2,]    1    6
```

```
solve(R)
          [,1]      [,2]
[1,]  0.2727 -0.0909
[2,] -0.0455  0.1818
```

1.5 Statistical Data Types

1.5.1 Data Types

To use statistical learning methods correctly, it is very important to understand the classification of the types of data that exist. This is of paramount importance because data are the input to all statistical machine learning methods and because the data type *determines* the appropriate and valid analysis to be implemented; in addition, each statistical machine learning method is specific to a certain type of data. In general, data are most commonly classified as quantitative (numerical) or qualitative (categorical) (Fig. 1.4).

By quantitative (numerical) data, we understand that the result of the observation or the result of a measurement is a number. They are classified as

(a) Discrete. The variable can only have point values and no values in between, that is, the variable can only have a certain set of possible values and represent items that can be counted because they only have isolated numerical values. Examples: number of household members, number of surgical interventions, number of reported cases of a certain pathology, number of accidents per month, etc. Examples in the context of plant breeding are panicle number per plant, seed number per panicle, weed count per plot, number of infected spikelets per spike, etc. Also, discrete values are called as count responses and those models based on Poisson and negative binomial distribution are appropriate for this type of responses.

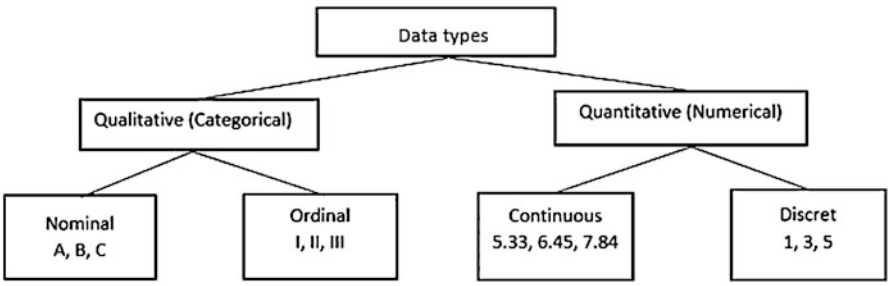

Fig. 1.4 Types of data

(b) Continuous. They are usually the result of a measurement that is expressed in particular units, and values are measured based on a zero point and are treated as real numbers. There are many types of mathematical operations that can be performed on this type of data. The measurements can theoretically have an infinite set of possible values within a range and they do not need transformation. In practice, the possible values of the variable are limited by the accuracy of the measurement method or by the recording mode. Examples: plant height, age, weight, grain yield, pH, blood cholesterol level, etc. The distinction between discrete and continuous data is important for deciding which statistical learning method to use for the analysis, since there are methods that assume that the data are continuous. Consider, for example, the age variable. Age is continuous, but if it is recorded in years, it turns out to be discrete. In studies with adults, in which the age ranges from 20 to 70 years, for example, there are no problems in treating age as continuous, since the number of possible values is large. But in the case of preschool children, if the age is recorded in years, it should be treated as discrete, while if it is recorded in months, it can be treated as continuous.

Similarly, the variable number of beats per minute is a discrete variable, but it is treated as continuous due to the large number of possible values. Numerical data (discrete or continuous) can be transformed into categorical and be treated as such. Although this is correct, it is not necessarily efficient because information is lost during the categorization process. It is always preferable to record the numerical value of the measurement, since this makes it possible to (a) analyze the variable as numerical because statistical analysis is simpler and more powerful and (b) form new categories using different criteria. Only in special cases it is preferable to record numerical data as categorical, for example, when the measurement is known to be imprecise (number of cigarettes per day, number of cups of coffee per week).

Categorical variables result from registering the presence of an attribute. The categories of a qualitative variable must be clearly defined during the design stage of the research and must be mutually exclusive and exhaustive. This means that each observation unit must be classified unambiguously in one, and only one, of the possible categories and that there is a category to classify each individual. In this sense, it is important to take into account all possibilities when constructing categorical variables, including a category such as “Do not know/No answer,” or “Not Registered,” or “Other,” which ensures that all the observed individuals will be classified based on the criteria that define the variable. Categorical data are also classified as (a) dichotomous, (b) nominal, and (c) ordinal.

(a) Two categories (dichotomous). The individual or observation unit can be assigned to only one of two categories. In general, it is about the presence or absence of the attribute and it is advantageous to assign code 0 to the absence and 1 to the presence.

(b) Examples: (1) resistance—no resistance, (2) disease—no disease, (3) tall—not tall, and (4) red color—no red color. It should be noted that examples 1 and 2 definitely cover all categories, while 3 and 4 are simplifications of more complex categories. In 3 and 4 it was necessary to establish a cutoff criterion to assemble a categorical variable from a numerical variable.
(c) More than two categories. When there are more than two categories, data can be nominal or ordinal. In nominal categories, there is no obvious order between the categories. These types of data values are distinct symbols, and these values serve as labels. The term "nominal" comes from the latin word for "name." Nominal attributes or labels have no relation to one another, nor is any order implied (Patterson and Gibson 2017). Some examples are religion: Catholicism, Islam, Judaism, etc.; race type: African, American, European, Asian, other; type of species; location; plant color; etc. In ordinal data, there is an obvious order between categories. Ordinal values have rank, giving us a notion of order but no concept of distance between the values. We can compare ordinal values with one another, but mathematical operations don't make sense in the context of these values (Patterson and Gibson 2017). Some examples are

1. Drought resistance: no resistance/low resistance/medium resistance/high resistance/total resistance
2. Disease severity: absent/mild/moderate/severe
3. Temperature of a process: hot/mild/cool
4. Social class: lower/middle/upper

Even when ordinal data can be coded as numbers as in the case of stages of drought resistance from 1 to 5 (1 = no resistance, 2 = low resistance, 3 = medium resistance, 4 = high resistance, 5 = total resistance), we cannot say that a plant in stage 4 has a drought resistance twice as strong as the resistance of a plant in stage 2, nor that the difference between stages 1 and 2 is the same as between stages 3 and 4. In contrast, when considering the age of a person, 40 years is twice 20 and a difference of 1 year is the same across the entire range of values. Therefore, we need to be aware that in ordinal data the difference between categories does not make sense.

Ordinal traits are very common in plant breeding programs for measuring disease incidence and severity and for sensory evaluation, such as the perceived quality of a product (e.g., taste, smell, color, decay) and plant development (e.g., developmental stages, maturity). These types of data are often partially subjective since the scale indicates only relative order and no absolute amounts; therefore, the intervals between successive categories may not be the same (Simko and Piepho 2011).

For this reason, we must be careful when dealing with qualitative variables, especially when they have been coded numerically, since they cannot be analyzed as numbers but must be analyzed as categories. It is incorrect to present, for example, the average stage of drought resistance in a group of plants.

In practice, scales are used to define degrees of a symptom or a disease, such as 1, 2, 3, 4, and 5. For this reason, it is important to operationally define this type of variables and study their reliability in order to ensure that two observers placed in front of the same plant will classify it in the same category.

Table 1.2 Examples of multivariate data

Units	Variables	Types of data
Plant	Several measurements of plant height on a single plant in time	All continuous
Animal	Measurement of three animal traits (average daily weight gain, muscularity, and calving)	Mixture of continuous and ordinal
Students	Grades in mathematics, physics, chemistry, biology	All continuous
People	Income, type of residence, gender, educational level, occupation	Mixture of nominal, ordinal, and continuous
Wheat plant	Measurement of four traits: grain yield, panicle number per plant, drought resistance, and type of wheat (common or durum)	Mixture of continuous, discrete, ordinal, and nominal
Country	Several measurements of school performance using the programme for international student assessment (PISA) test	Exam scores continuous in mathematics, reading and science

1.5.2 Multivariate Data Types

Practitioners and researchers in all applied disciplines often measure several variables in each observation, subject, unit, or experimental unit. That is, all variables are simultaneously measured in the same observation. Multivariate data are very common in all disciplines due to the need and facility for data collection in most fields. These variables can consist of only one type of data (for example, plant height measured using a continuous scale on each plant 12 times every 15 days) or a mixture of data types, for example, measuring, on each plant, four different traits: grain yield (on a continuous scale), disease resistance (on an ordinal scale), flower color (nominal scale), and days to flowering (discrete or count). Table 1.2 provides other examples of multivariate data measured using only one scale or a mixture of scales.

It is important to point out that here all measurements are done simultaneously in each observation. For this reason, they are classified as multivariate type of data and include data that will be used as dependent variables or independent variables in the process of training the statistical machine learning algorithms that will be studied here.

1.6 Types of Learning

The three most common ways of learning in statistical machine learning are (a) supervised learning, (b) unsupervised learning, and (c) semi-supervised learning. The three methods are explained below.

1.6.1 Definition and Examples of Supervised Learning

Supervised learning can be defined as the process of learning a function that maps an input to an output based on teaching the statistical machine learning method with input–output pairs. The training data consist of pairs of objects (usually vectors): one component of the pair is the input data (predictors = explanatory variable = input) and the other, the desired results (response variable = dependent variable = output). The output of the function can be a numerical value (as in regression problems) or a class label (as in multinomial regression). The goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data. This function should be capable of predicting the value corresponding to any valid input object after having seen a series of examples of training data. Under optimal conditions, the algorithm correctly determines the class labels for unseen instances. This implies a learning algorithm that is able to generalize from the training data to unseen situations in a "reasonable" way.

Suppose you're teaching your child to distinguish between corn and tomato (Fig. 1.5). First you show him (her) a picture of an ear of corn and a picture of a tomato. In the learning process, your child must keep in mind that if the color is yellow and the shape is not round, then it is probably an ear of corn, but if the color is red and the shape is round, then it is probably a tomato. This is how your child learns. Then you can show a third picture and ask your child to classify the vegetable as either ear of corn or tomato. When you show the third picture, he (she) will very likely identify if the vegetable is ear of corn or tomato, due to the fact that we have already labeled the two pictures into categories, so your child knows what is an ear of

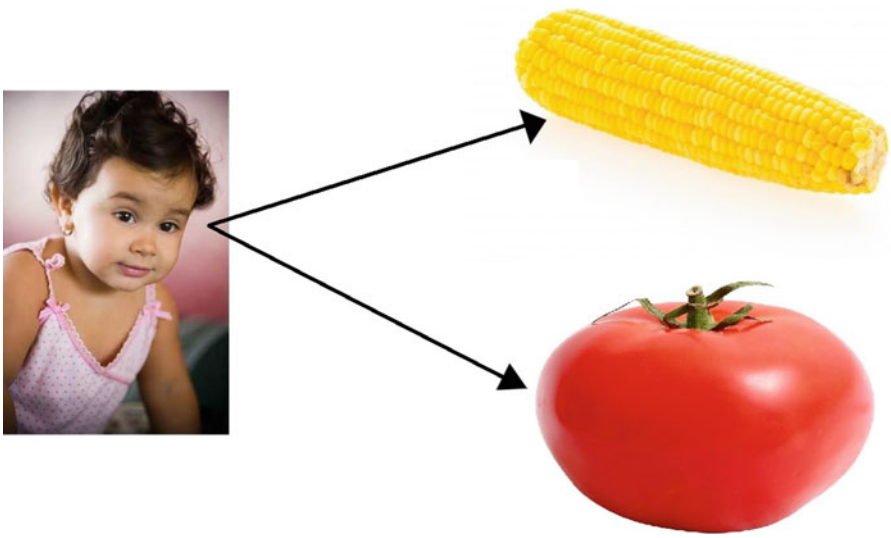

Fig. 1.5 Supervised learning process for teaching a child to distinguish tomato from corn

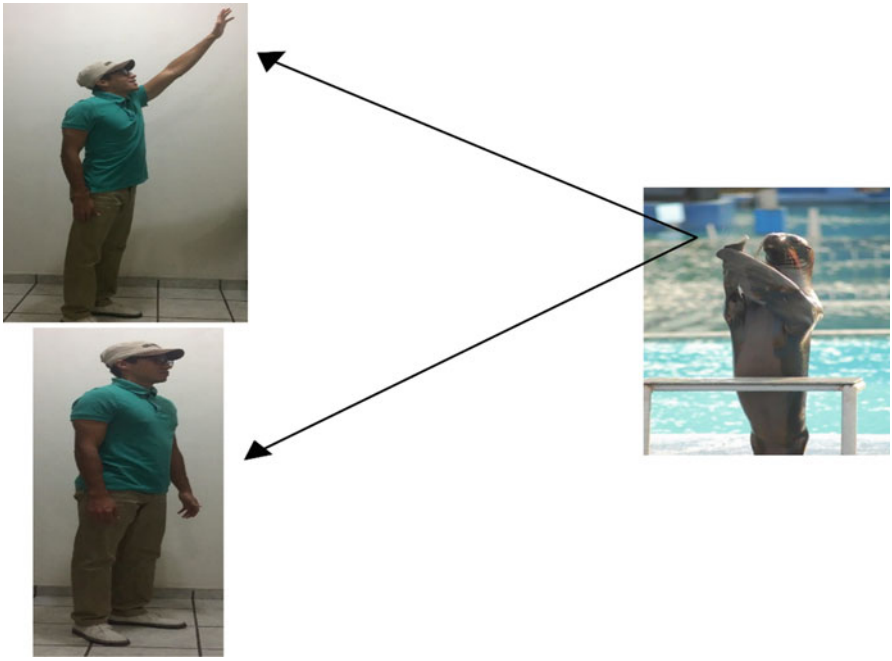

Fig. 1.6 Supervised learning process for teaching a seal to applaud

corn and what is a tomato. This example illustrates how supervised learning works using ground truth data that consist of having prior knowledge of what the output values of our samples should be.

To give another example, imagine that you're training a seal to applaud (Fig. 1.6). The goal is to make the seal applaud when you raise your right hand. The training process consists of presenting the seal with enough examples by raising your right hand and rewarding it with some great candy whenever it applauds when it sees your right hand is raised. In the same way, the seal may be "punished" if it applauds whenever your right hand is not raised, by doing something unpleasant for the seal but not harmful. Supervision involves stimulating the seal to respond to positive samples by rewarding it, and not to respond to negative samples by "punishing" it. Hopefully, the seal then obtains a built-in feeling (hypothesis) for applauding whenever you raise your hand right. The process is evaluated by presenting the seal with another person raising his/her right hand, someone who did not take part in the training process and who is unknown to the seal. However, based on its built-in feeling for what a person with his/her raised right hand looks like, the seal should be able to transfer this knowledge to the present person. It must then consider and decide whether or not it wants to signal the presence of a right hand raised by applauding.

Next we provide some real examples. In the first example, a scientist has thousands of molecules and information about which ones are drugs and he trains

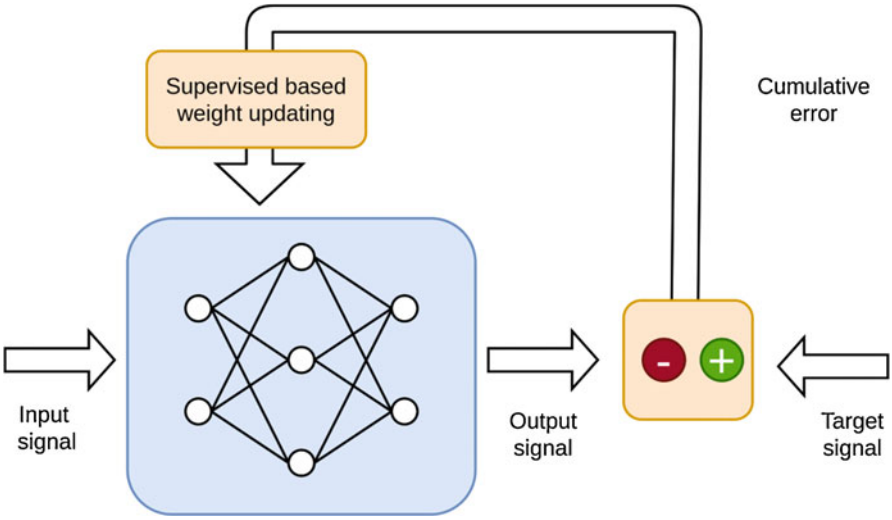

Fig. 1.7 Diagram illustrating a supervised learning process where there are inputs (X) and response variables (y) for each observation

a statistical machine learning model to determine whether a new molecule is also a drug. In the context of plant breeding, a scientist collects hundreds of markers (genetic information) and thousands of images of hundreds of plants. For this sample of plants, he also measures their phenotype (grain yield) because he wants to implement a statistical machine learning algorithm to estimate (predict) the grain yield of new plants not used in the training process. Another example is in environmental science, where scientists use historical data (as when it's sunny and the temperature is higher, or when it's cloudy and the humidity is higher, etc.) to train a statistical machine learning model to predict the weather for a given future time.

In a more mathematical way, under supervised learning, we usually have access to a set of p predictor (input) variables $X_1, X_2, \ldots, X_p$ measured in n observations, and a response variable (output) Y also measured in those same n observations (Fig. 1.7). The goal is to predict Y using a function of $X_1, X_2, \ldots, X_p$, that is, we use an algorithm to learn the mapping function from the input to the output $Y = f(X_1, X_2, \ldots, X_p)$, and we expect to estimate the mapping function so well that when we have new input data, we can predict the output variables for those data. The term supervised learning was coined because the learning process of any statistical machine learning method from the training dataset can be thought of as a teacher supervising the learning process. We know the correct outputs (response variables), the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the statistical machine learning algorithm achieves an acceptable level of performance.

1.6.2 *Definitions and Examples of Unsupervised Learning*

Unsupervised learning is when you only have input (predictors = independent variables) data (X) and no previous knowledge of corresponding labeled outputs or response variables (Fig. 1.8). So its goal is to deduce the natural structure present within a set of data points. In other words, to extract the underlying structure or distribution in the data in order to learn more about the data, that is, the network uses training patterns to discover emerging collective properties and organizes the data into clusters. In unsupervised learning (unlike supervised learning), there is no correct answer (output = response variable = dependent variable) and there is no teacher. For this reason, we are not interested in prediction since we do not have an associated response variable Y. Statistical machine learning algorithms under unsupervised learning are left to their own devices to discover and present the interesting structure in the data. However, there is no way to determine if our work is correct since we don't know the right answer because the job was done without supervision. Unsupervised learning problems can be divided into clustering and association problems.

Clustering: A clustering problem is when you want to discover the inherent groupings in the data, such as grouping maize hybrids by their genetic architecture. Another example is grouping people according to their consumption behaviors. But in both cases we cannot check if the classifications are correct since we don't know the true grouping of each individual.

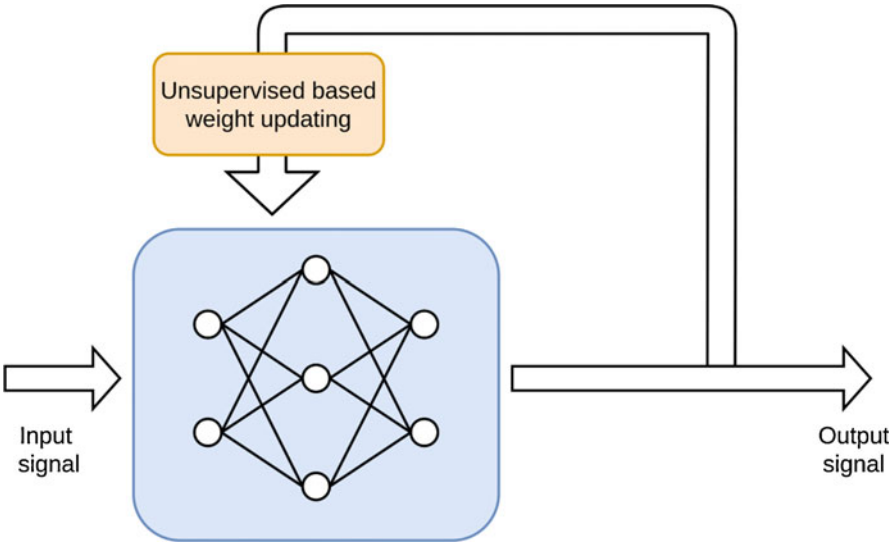

Fig. 1.8 Diagram illustrating an unsupervised learning process where there are only inputs (X), but no response variables (y) for each observation

Association: An association rule learning problem is when you want to discover rules that describe large portions of your data, such as people who buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are

(a) Principal component analysis
(b) Multidimensional scaling for grouping
(c) A k-means algorithm for clustering problems
(d) An a priori algorithm for association rule learning problems

1.6.3 Definition and Examples of Semi-Supervised Learning

Semi-supervised learning problems are those that have a large amount of input data (X) available but only some of the data are labeled (Y). For this reason, these problems are positioned between supervised and unsupervised learning. A good example is plant species classification using thousands of images where only some of the images are labeled (e.g., species 1, species 2, species 3, etc.) and the majority are unlabeled. Another example is the classification of exoplanets (exoplanets are planets that are outside our solar system) also using thousands of photos where only a small fraction of the photos is labeled (four types of exoplanets). Many real-world problems in the context of statistical machine learning belong to this type of learning process. This is because it is more expensive and time-consuming to use labeled data than unlabeled data since many times this requires having access to domain experts, whereas it is cheap and easy to collect and store unlabeled data.

References

Adler I (1912) Primary malignant growths of the lungs and bronchi: a pathological and clinical study. Longmans, Green and Co, New York and London, p 325

Bernardo R (2016) Bandwagons I, too, have known. Theor Appl Genet. https://doi.org/10.1007/s00122-016-2772-5

Box GEP (1976) Science and statistics (PDF). J Am Stat Assoc 71:791–799. https://doi.org/10.1080/01621459.1976.10480949

Breiman L (2001) Statistical modeling: the two cultures. Stat Sci 16:199–215

Crossa J, Pérez-Rodríguez P, Cuevas J, Montesinos-López OA, Jarquín D, de Los Campos G, Burgueño J, González-Camacho JM, Pérez-Elizalde S, Beyene Y, Dreisigacker S, Singh R, Zhang X, Gowda M, Roorkiwal M, Rutkoski J, Varshney RK (2017) Genomic selection in plant breeding: methods, models, and perspectives. Trends Plant Sci 22(11):961–975

Dean J (2018) Big data, data mining, and machine learning. Value creation for business leaders and practitioners. John Wiley & Sons, Inc., Hoboken

FAO (2011) The state of the World's land and water resources for food and agriculture: managing Systems at Risk. Food and agriculture Organization of the United Nations. FAO, Rome

Fischer T, Byerlee D, Edmeades G (2014) Crop yields and global food security. ACIAR, Canberra

James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning: with applications in R. Springer, New York

McKinsey Global Institute (2016) The age of analytics: competing in a data-driven world. https://www.mckinsey.com/~/media/mckinsey/business%20functions/mckinsey%20analytics/our%20insights/the%20age%20of%20analytics%20competing%20in%20a%20data%20driven%20world/mgi-the-age-of-analytics-executive-summary.ashx

Meuwissen THE, Hayes BJ, Goddard ME (2001) Prediction of total genetic value using genome-wide dense marker maps. Genetics 157:1819–1829

Milliken GA, Johnson DE (2009) Analysis messy of data, volume 1 designed experiments. CRC Press Taylor & Francis Group, Boca Raton, London, New York

Montesinos-López A, Martín-Vallejo J, Crossa J, Gianola D, Hernández-Suárez CM, Montesinos-López OA, Juliana P, Singh R (2019) A benchmarking between deep learning, support vector machine and Bayesian threshold best linear unbiased prediction for predicting ordinal traits in plant breeding. G3 9(2):601–618

Oury F-X, Godin C, Mailliard A, Chassin A, Gardet O, Giraud A, Heumez E, Morlais J-Y, Rolland B, Rousset M, Trottet M, Charmet G (2012) A study of genetic progress due to selection reveals a negative effect of climate change on bread wheat yield in France. Eur J Agron 40:28–38

Patterson J, Gibson A (2017) Deep learning: a Practitioner's approach. O'Reilly Media, Beijing

Pinheiro JC, Bates DM (2000) Mixed-effects models in S and S-PLUS. Springer Verlag, New York

Proctor RN (2012) The history of the discovery of the cigarette-lung cancer link: evidentiary traditions, corporate denial, global toll. Tob Control 21(2):87–91

Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3 (3):210–229

Schuster PM (2014) The scientific life of Victor Franz (Francis) Hess (June 24, 1883–December 17, 1964). Astropart Phys 53:33–49

Sejnowski TJ (2018) The deep learning revolution. The MIT Press, Cambridge, MA, London

Shmueli G (2012) To explain or to predict? Stat Sci 25(3):289–310. https://doi.org/10.1214/10-STS330

Simko I, Piepho H-P (2011) Combining phenotypic data from ordinal rating scales in multiple plant experiments. Trends Plant Sci 16:235–237

Stroup W (2012) Generalized linear mixed models: modern concepts, methods and applications. CRC Press, Boca Raton

Wang X, Xua Y, Hu Z, Hu C (2018) Genomic selection methods for crop improvement: current status and prospects. Crop J 6(4):330–340

Wolpert DH (1996) The lack of a priory distinction between learning algorithms. Neural Comput 8 (7):1341–1390

Chapter 2
Preprocessing Tools for Data Preparation

2.1 Fixed or Random Effects

As mentioned in Chap. 1, fixed effect models in general (to design experiments, regression models, genomic prediction models, etc.) are recommended when the levels under study (collected by the scientist) are the unique levels of interest in the study, and the levels or quantities observed in the explanatory variables are treated as if they were nonrandom. For these reasons, a fixed factor is defined as a categorical or classification variable, chosen to represent specific conditions, for which the researcher has included all levels (or conditions) that are of interest for the study in the model. This means that fixed effects are unknown constant parameters associated with continuous covariates or levels of categorical factors in any fixed or mixed effects (fixed + random effects). The estimation of these fixed parameters in fixed effects models or mixed effects models is generally of intrinsic interest, since they indicate the relationships of the covariates with the response variable. Fixed effects can be associated with continuous covariates such as the weight of an animal in kilograms, maize yield in tons per hectare, qualification of a reference test or socioeconomic level, which will carry a continuous range of values, or they can be associated with factors such as gender, hybrid, or group treatment, which are categorical. This implies that fixed effects are the best option when performing an inference for the whole target population.

A random factor is a classification variable with levels that can be randomly sampled from a population with different levels of study, such as classrooms, regions, cattle herds, or clinics that are randomly sampled from a population. All possible levels of the random factor are not present in the data set, yet it is the intention of the researcher to make an inference about the entire population of levels from the selected sample of these factor levels. Random factors are included in an analysis in order for the modification in the dependent variable through the levels of the random factors to be evaluated and the results of the data analysis generalized to all levels of the population random factor. This means that random effects are

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_2

represented by random variables (not observed) which, we generally assume, have a particular distribution, the normal distribution being the most common. Due to the above, random effects are suggested when we want to perform an inference for all levels of the target population.

2.2 BLUEs and BLUPs

This section presents the concepts and terminologies of BLUE and BLUP. Since these two concepts are related to a mixed model, we present the following linear mixed model as

$$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{Zu} + \boldsymbol{\varepsilon}, \tag{2.1}$$

where $\boldsymbol{Y}$ is the vector of response variables of order $n \times 1$, $\boldsymbol{X}$ is the design matrix of fixed effects of order $n \times p$, $\boldsymbol{\beta}$ is the vector of order $p \times 1$ of beta coefficients, $\boldsymbol{Z}$ is the design matrix of random effects of order $n \times q$, $\boldsymbol{u}$ is the vector of random effects distributed as $\boldsymbol{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is a variance–covariance matrix of random effects of dimension $q \times q$, and $\boldsymbol{\varepsilon}$ is a vector of residuals distributed as $\boldsymbol{N}(\boldsymbol{0}, \boldsymbol{R})$, where $\boldsymbol{R}$ is a variance–covariance matrix of residual effects of dimension $n \times n$. The unconditional mean of $\boldsymbol{Y}$ is equal to $E(\boldsymbol{Y}) = \boldsymbol{X\beta}$, while the conditional mean of $\boldsymbol{Y}$, given the random effects, is equal to $E(\boldsymbol{Y}|\boldsymbol{u}) = \boldsymbol{X\beta} + \boldsymbol{Zu}$. A solution to jointly "estimate" parameters $\boldsymbol{\beta}$ and $\boldsymbol{u}$ was proposed by Henderson (1950, 1963, 1973, 1975, 1984), which consists in solving the mixed model equation (MME)

$$\begin{pmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{Z} \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{Z} + \boldsymbol{\Sigma}^{-1} \end{pmatrix} \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\boldsymbol{u}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \end{pmatrix} \tag{2.2}$$

The solution obtained for $\boldsymbol{\beta}$ is the BLUE and the solution obtained for $\boldsymbol{u}$ is the BLUP.

While this expression to find the estimates of $\widehat{\boldsymbol{\beta}}$ and $\widehat{\boldsymbol{u}}$ may look quite complex, when the number of observations is larger than the sum of the number of fixed effects and the number of random effects $(p + q)$, it is quite efficient since only needs to calculate the inverse of the small matrices of $\boldsymbol{R}$ and $\boldsymbol{\Sigma}$. Also, the matrix on the left that needs to be inverted to obtain the solution for $\widehat{\boldsymbol{\beta}}$ and $\widehat{\boldsymbol{u}}$ is of order $(p + q) \times (p + q)$, which in some applications is considerably less than a matrix of dimension $n \times n$ as $\boldsymbol{V} = \boldsymbol{Z\Sigma Z}^{\mathrm{T}} + \boldsymbol{R}$, which is also useful to obtain the solution of these parameters by using $\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}$ and $\widehat{\boldsymbol{u}} = \boldsymbol{\Sigma Z}^{\mathrm{T}}\left(\boldsymbol{y} - \boldsymbol{X}\widehat{\boldsymbol{\beta}}\right)$. Under both solutions, for $\widehat{\boldsymbol{\beta}}$ and $\widehat{\boldsymbol{u}}$ it is assumed that the covariance matrices are known, but in practice these are replaced by estimations and the results are known as empirical BLUE (EBLUE) and empirical BLUP (EBLUP).

The linear combinations of the parameters are called **estimable functions** if they can be constructed from a linear combination of unconditional means (of fixed effects only) of the observations (Littell et al. 1996). Estimable functions do not depend on random effects. Below, we provide a formal definition of an estimable function.

Definition of an estimable function $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta}$ is estimable if there is a matrix $\boldsymbol{T}$ such that

$$\boldsymbol{T}^{\mathrm{T}}\boldsymbol{E}(\boldsymbol{Y}) = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta} \; \forall \, \boldsymbol{\beta}.$$

One way of testing candidate matrices for estimability is to use the following result:

$$\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta} \text{ is estimable if, and only if, } \boldsymbol{K}^{\mathrm{T}}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right) = \boldsymbol{K}^{\mathrm{T}},$$

where $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-}$ denotes a generalized inverse of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. Quantities such as regression coefficients, treatment means, treatment differences, contrasts, and simple effects in factorial experiments are all common examples of estimable functions and their resulting estimates are examples of BLUEs (Littell et al. 2006). BLUEs correspond to broad inference because they are valid for the whole population under study, and are also called population average inference using the terminology by Zeger et al. (1998). Table 2.1 presents predictors of some regression models and some common experimental designs and also provides some functions of the predictor that are and are not estimable functions.

Table 2.1 indicates that estimable functions used to obtain BLUEs are only linear combinations of fixed effects, and the inference focuses on the average performance throughout the target population. Although there are many possible linear combinations of fixed effects that can be of interest to estimate with BLUEs, the most important ones are treatment means expressed as $\eta_0 + \tau_i$, the difference between treatments $\tau_i - \tau_{i'}$ and simple effects. In general, the most common BLUEs can be obtained from any fitted generalized mixed model using the expression $\text{BLUE} = g^{-1}\left(\boldsymbol{X}\widehat{\boldsymbol{\beta}}\right)$, where $g^{-1}(\cdot)$ is the inverse link used to fit the generalized mixed model and the inference is related to population-wide average (broad).

Estimability matters because many models are not full rank, such as analysis of variance (ANOVA) models, where estimating equation solutions for the effects themselves has no intrinsic meaning and solutions depend entirely on the generalized inverse used. Theory says that there is an infinite number of ways to construct a generalized inverse. While estimable functions are invariant of generalized inverse and therefore have an assignable meaning, that is, although the effect estimates per se do not have any legitimate interpretation, estimable functions do (Stroup 2012). Next, an example of how to obtain the BLUEs of genotypes (treatments) under a randomized complete block design is provided.

Table 2.1 Predictors and some estimable and non-estimable functions of each predictor of some common useful linear models

Model	Predictor	Estimable functions	Non-estimable functions
Complete randomized design (CRD)	$\eta_i = \eta_0 + \tau_i$	$\eta_0 + \tau_i$, $\tau_i - \tau_{i'}$, $i \neq i'$	τ_i, η_0
Randomized complete blocks (RCBD)	$\eta_{ij} = \eta_0 + \tau_i + b_j$, $b_j \sim N(0, \sigma_b^2)$	$\eta_0 + \tau_i$, $\tau_i - \tau_{i'}$, $i \neq i'$	τ_i, η_0
Regression	$\eta_i = \eta_0 + \sum_{j=1}^{p} x_{ij}\beta_j$	$\eta_0 + \sum_{j=1}^{p} x_{ij}\beta_j$, $\eta_0 + x_{ij}\beta_j$	η_0
Split plot design in a CRD	$\eta_{ijk} = \eta_0 + \alpha_i + \alpha(r)_{ik} + \beta_j + (\alpha\beta)_{ij}$; $\beta_j \sim N\left(0, \sigma_\beta^2\right)$, $(\alpha\beta)_{ij} \sim N\left(0, \sigma_{\alpha\beta}^2\right)$, $\alpha(r)_{ik} \sim N\left(0, \sigma_{\alpha(r)}^2\right)$	$\eta_0 + \alpha_i$	η_0
Split plot design in RCBD	$\eta_{ijk} = \eta_0 + \alpha_i + r_k + \alpha(r)_{ik} + \beta_j + (\alpha\beta)_{ij}$; $\beta_j \sim N\left(0, \sigma_\beta^2\right)$, $(\alpha\beta)_{ij} \sim N\left(0, \sigma_{\alpha\beta}^2\right)$, $r_k \sim N(0, \sigma_r^2)$ $\alpha(r)_{ik} \sim N\left(0, \sigma_{\alpha(r)}^2\right)$	$\eta_0 + \alpha_i$	η_0

Example 1 Grain yield of five genotypes evaluated in a randomized complete block design. The data of this experiment are shown in Table 2.2. This example is provided to illustrate the process of estimating the BLUEs of genotypes.

Next, we provide the R code that uses the lme4 library to fit a mixed model for the data given in Table 2.2 to estimate the BLUEs of genotypes:

```
Data_RCBD=read.table("Example_RCBD.csv", header =T, sep = ",")
Data_RCBD

library(lme4)
Data_RCBD$Genotype=as.factor(Data_RCBD$Genotype)
Data_RCBD$Block=as.factor(Data_RCBD$Block)

Fitted=lmer(Yield~ Genotype + (1 | Block), Data_RCBD)
Fitted
####Extracting design matrix of fixed effects (Intercept and Genotype)
X=Fitted@pp$X
X=X[!duplicated(X), ]
X
####Extracting the beta coefficients
Beta=Fitted@beta #fixef(Fitted)
####Obtaining the BLUEs of genotypes
BLUEs_Gen=X%*%Beta
BLUEs_Gen
```

Table 2.2 Grain yield (Yield) of five genotypes under a randomized complete block design

Block	Genotype	Yield
1	1	5.25
1	2	9
1	3	6.25
1	4	4.5
1	5	5.5
2	1	6.5
2	2	9.5
2	3	6.75
2	4	4.25
2	5	6.5
3	1	4
3	2	6.25
3	3	5.5
3	4	4.5
3	5	5.25
4	1	7
4	2	8.75
4	3	6.75
4	4	5
4	5	6

The above code shows that Genotype was specified as a fixed effect, although Block was specified as a random effect. It is important to point out that both effects were converted to factors. The output of this fitted model is given below.

```
> Fitted
Linear mixed model fit by REML ['lmerMod']
Formula: Yield ~ Genotype + (1 | Block)
  Data: Data_RCBD
REML criterion at convergence: 43.1701
Random effects:
 Groups  Name        Std.Dev.
 Block   (Intercept) 0.6922
 Residual            0.6739
Number of obs: 20, groups: Block, 4
Fixed Effects:
(Intercept)  Genotype2  Genotype3  Genotype4  Genotype5
   5.688    2.688    0.625    -1.125    0.125
> ####Extracting design matrix
> X=Fitted@pp$X
> X=X[!duplicated(X), ]
> X
  (Intercept) Genotype2 Genotype3 Genotype4 Genotype5
1           1         0         0         0         0
5           1         1         0         0         0
9           1         0         1         0         0
13          1         0         0         1         0
17          1         0         0         0         1
```

```
> ####Extracting the fixed effects
> Beta=Fitted@beta
> ####Obtaining the BLUEs of genotypes
> BLUEs_Gen=X%*%Beta
> BLUEs_Gen
     [,1]
1   5.6875
5   8.3750
9   6.3125
13  4.5625
17  5.8125
```

The above code shows that the standard deviation of the random effect of blocks was equal to 0.6922, while the standard deviation of the residual was equal to 0.6739. In the part that reads "Fixed Effects" we find the beta coefficient estimates of the fixed effects with which the BLUEs of each of the genotypes can be obtained. With the Fitted@pp$X, the design matrix of fixed effects is extracted as implemented in lmer, and with X[!duplicated(X),] the rows which are duplicated due to blocks being removed. Then, with Fitted@beta, the beta coefficients for the fixed effects are extracted and, finally, with X%*%Beta, the BLUEs of each genotype are obtained. Since the estimable function (Table 2.1) for genotypes (treatments) are $\eta_0 + \tau_i$, the BLUEs of each genotype are computed as $5.688 + 0 = 5.688$ (BLUE of genotype 1), $5.688 + 2.688 = 8.376$ (BLUE of genotype 2), $5.688 + 0.625 = 6.313$ (BLUE of genotype 3), $5.688 - 1.125 = 4.563$ (BLUE of genotype 4), and $5.688 + 0.125 = 5.813$ (BLUE of genotype 5). Since for more complex models, obtaining the BLUEs for genotypes can be quite laborious, these can be obtained directly using the following lines of code:

```
library(lsmeans)
Lsmeans_Gen=lsmeans(Fitted,~ Genotype)
#Lsmeans_Gen
BLUEs_Gen=data.frame(GID=Lsmeans_Gen$Genotype,
          BLUEs=Lsmeans_Gen$lsmean,
          SE_BLUEs=Lsmeans_Gen$SE)
BLUEs_Gen
```

In this way we get

```
> BLUEs_Gen
 GID BLUEs SE_BLUEs
1  1 5.6875 0.4830459
2  2 8.3750 0.4830459
3  3 6.3125 0.4830459
4  4 4.5625 0.4830459
5  5 5.8125 0.4830459
```

Predictable functions These are linear combinations of the fixed and random effects, $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta} + \boldsymbol{M}^{\mathrm{T}}\boldsymbol{u}$, that is, they can be formed from linear combinations of the conditional means: $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta} + \boldsymbol{M}^{\mathrm{T}}\boldsymbol{u}$ is a predictable function if $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta}$ is estimable. The inference based on these predictable functions is referred to as narrow inference, which, unlike broad inference, has random effects such as additional terms and limits the attention to a group of the sampled random levels (Littell et al. 2006). Then, replacing the estimates obtained from the mixed model equation (2.2) in predictable functions, $\boldsymbol{K}^{\mathrm{T}}\widehat{\boldsymbol{\beta}} + \boldsymbol{M}^{\mathrm{T}}\widehat{\boldsymbol{u}}$ results in the best linear unbiased predictors (BLUPs) of the corresponding predictable function. From a theoretical point of view, BLUP is expected to have a better genotypic predictive accuracy than BLUE, which is important for the selection of new cultivars, or even the genetic values (additive effects) for the selection of progenitors (Piepho et al. 2008). Before BLUP-based selection, selection in crop breeding was based on either simple arithmetic means or BLUEs of genotypes, which can also be calculated in a mixed model context based on fixed genotype effects (Piepho et al. 2008).

BLUP has a long tradition of selection in animal science, but its use in plant breeding is very recent. It is therefore important to provide a clear distinction between the two terms.

Table 2.3 presents the corresponding predictable functions for the same models described in Table 2.1.

It is important to point out that BLUEs and BLUPs are not restricted only to linear mixed models (Eq. 2.1), because they can be obtained in an approximate manner for

Table 2.3 Some predictable functions for the same models given in Table 2.1

Model	Predictor	Predictable function
Complete randomized design (CRD)	$\eta_i = \eta_0 + \tau_i$	None, since there are no random effects
Randomized complete blocks (RCBD)	$\eta_{ij} = \eta_0 + \tau_i + b_j$, $b_j \sim N\left(0, \sigma_b^2\right)$	$\eta_0 + \tau_i + b_j$
Regression	$\eta_i = \eta_0 + \sum_{j=1}^{p} x_{ij}\beta_j$	None, since there are no random effects
Split plot design in a CRD	$\eta_{ijk} = \eta_0 + \alpha_i + \alpha(r)_{ik} + \beta_j + (\alpha\beta)_{ij}$; $\beta_j \sim N\left(0, \sigma_\beta^2\right)$, $(\alpha\beta)_{ij} \sim N\left(0, \sigma_{\alpha\beta}^2\right)$, $\alpha(r)_{ik} \sim N\left(0, \sigma_{\alpha(r)}^2\right)$	$\eta_0 + \alpha_i + \beta_j + (\alpha\beta)_{ij}$
Split plot design in RCBD	$\eta_{ijk} = \eta_0 + \alpha_i + r_k + \alpha(r)_{ik} + \beta_j + (\alpha\beta)_{ij}$; $\beta_j \sim N\left(0, \sigma_\beta^2\right)$, $(\alpha\beta)_{ij} \sim N\left(0, \sigma_{\alpha\beta}^2\right)$, $r_k \sim N\left(0, \sigma_r^2\right)$ $\alpha(r)_{ik} \sim N\left(0, \sigma_{\alpha(r)}^2\right)$	$\eta_0 + \alpha_i + \beta_j + (\alpha\beta)_{ij}$

Table 2.4 BLUEs for mean response and BLUPs for conditional mean response for different types of response variables

Type of response variable	Distribution	Link function	BLUE	BLUP
Continuous	Normal	Identity	$\boldsymbol{X\hat{\beta}}$	$\boldsymbol{X\hat{\beta}+Z\hat{u}}$
Binary	Binomial	Logit	$\frac{1}{1+\exp\left(-\boldsymbol{X\hat{\beta}}\right)}$	$\frac{1}{1+\exp\left(-\boldsymbol{X\hat{\beta}}-\boldsymbol{Z\hat{u}}\right)}$
Binary	Binomial	Probit	$\Phi\left(\boldsymbol{X\hat{\beta}}\right)$	$\Phi\left(\boldsymbol{X\hat{\beta}}+\boldsymbol{Z\hat{u}}\right)$
Counts	Poisson	Log	$\exp\left(\boldsymbol{X\hat{\beta}}\right)$	$\exp\left(\boldsymbol{X\hat{\beta}}+\boldsymbol{Z\hat{u}}\right)$
Counts	Negative binomial	Log	$\exp\left(\boldsymbol{X\hat{\beta}}\right)$	$\exp\left(\boldsymbol{X\hat{\beta}}+\boldsymbol{Z\hat{u}}\right)$
Continuous proportions	Beta	Logit	$\frac{1}{1+\exp\left(-\boldsymbol{X\hat{\beta}}\right)}$	$\frac{1}{1+\exp\left(-\boldsymbol{X\hat{\beta}}-\boldsymbol{Z\hat{u}}\right)}$
Continuous positives	Gamma	Inverse	$\frac{1}{\boldsymbol{X\hat{\beta}}}$	$\frac{1}{\boldsymbol{X\hat{\beta}}+\boldsymbol{Z\hat{u}}}$

Φ is the cumulative distribution function (CDF) of the standard normal distribution

any fitted generalized linear mixed model using the expression BLUP $= g^{-1}\left(\boldsymbol{X\hat{\beta}}\right.$ $\left.+\boldsymbol{Z\hat{u}}\right)$ (Stroup 2012). For example, Table 2.4 provides the BLUEs and BLUPs for some of the most popular response variables under a predictor with fixed and random effects.

In sum, the linear combinations of fixed effects only are called estimable functions and give rise to BLUEs. The solution of mixed model equations produces estimates, or BLUEs, for linear combinations of the form $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta}$. Linear combinations of fixed *and* random effects are called predictable functions. Solving the mixed model equations yields *predictors*, or BLUPs, which are used to obtain BLUPs of linear combinations such as $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{\beta}+\boldsymbol{M}^{\mathrm{T}}\boldsymbol{u}$. The best of both BLUEs and BLUPs means that these estimates have minimum mean square errors (see Searle et al. 2006) for the different meanings of the criteria applied to each one.

Both the BLUEs and BLUPs are not possible to be computed in real applications, since true variance–covariance values of $\boldsymbol{R}$ and $\boldsymbol{\Sigma}$ are required, but we only have access to estimates of these variance–covariance matrices. For this reason, only empirical BLUEs and empirical BLUPs are possible, since we use variance–covariance parameter estimates ($\widehat{\boldsymbol{R}}$ and $\widehat{\boldsymbol{\Sigma}}$) to solve the mixed model equations for $\boldsymbol{\beta}$ and $\mathbf{u}$.

Below, we illustrate the calculation of the BLUPs of genotypes of the data set given in Table 2.2 for which the BLUEs of genotypes were obtained. Using the lmer() function again, but now assuming that the genotype is a random effect, instead of fixed effects we obtained the following output of the fitted model. We can see that the standard deviation of genotypes is 1.3575, the standard deviation of blocks is 0.6922, the standard deviation of residuals is 0.6739, and the intercept is equal to 6.15.

```
> #####BLUP of genotypes
> Fitted2=lmer(Yield~ (1|Genotype) + (1 | Block), Data_RCBD)
> Fitted2
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: Yield ~ (1 | Genotype) + (1 | Block)
  Data: Data_RCBD
REML criterion at convergence: 58.8152
Random effects:
 Groups  Name     Std.Dev.
 Genotype (Intercept) 1.3575
 Block   (Intercept) 0.6922
 Residual       0.6739
Number of obs: 20, groups: Genotype, 5; Block, 4
Fixed Effects:
(Intercept)
    6.15
```

From the fitted model (Fitted2), we now extract the intercept with fixef(Fitted2) and the random effects of genotypes with c(ranef(Fitted2)$Genotype); then we sum up these two terms to get the BLUPs of genotypes that we called BLUP_Gen2, and finally, we calculate the correlation between the BLUPs and BLUEs (obtained above) for the same genotypes. This correlation was equal to one, which shows that we should not expect big differences between the use of BLUPs or BLUEs of genotypes, although there are large amounts of empirical evidence showing that the BLUPs should be preferred over the BLUEs and not always are the same results produced by both.

```
> #####Fixed effect=Intercept######
> Intercept=fixef(Fitted2)
> str(Intercept)
Named num 6.15
 - attr(*, "names")= chr "(Intercept)"
> #####Random effects of genoytpes
> U_ref=c(ranef(Fitted2)$Genotype)
> U_ref
$'(Intercept)'
[1] -0.4356567  2.0958619  0.1530686 -1.4953621 -0.3179116
> ########BLUP of Genotypes#####
> BLUP_Gen2=Intercept+U_ref$'(Intercept)'
> BLUP_Gen2
[1] 5.714343 8.245862 6.303069 4.654638 5.832088
> cor(c(BLUEs_Gen),BLUP_Gen2)
[1] 1
```

2.3 Marker Depuration

First, we will define markers and their importance. Markers are beneficial in the construction of precise genetic relationships, for parental determination and for the identification and mapping of quantitative trait loci (QTL). Between 1970 and 2001, most of the genetic progress in the livestock industry was reached by using pedigree

and phenotypic information. However, after the first draft of the human genome project was finished in 2001 (The International SNP Map Working Group 2001), the cost of genotyping using single nucleotide polymorphisms (SNPs) started to decrease considerably, and now its cost is at least 1000 times lower. For this reason, Stonecking (2001) points out that SNPs have become the bread and butter of DNA sequence variation and are essential in determining the genetic potential of livestock and plant breeding.

However, it is also important to point out that other types of DNA markers have been discovered, such as restriction fragment length polymorphisms (RFLP), simple sequence repeat (SSR), Diversity Arrays Technology (DArT), simple sequence length polymorphisms (SSLP), amplified fragment length polymorphisms (AFLP), etc. However, SNPs have become the main markers used to detect DNA variation for some of the following reasons: (a) SNPs are abundant and found throughout the entire genome, in intragenic and extragenic regions (Schork et al. 2000), (b) they represent the most common genetic variants, (c) the location in the DNA: they are found in introns, exons, promoters, enhancers, or intergenic regions, (d) they are easily evaluated by automated means, (e) many of them have direct repercussions on traits of interest in plant and animals, (f) they are generally biallelic, and (g) they are now cheap and easy to genotype.

It is important to remember that DNA (deoxyribonucleic acid) is organized in pairs of chromosomes, each inherited from one of the parents. The diversity found among organisms is a result of variations in DNA sequences and of environmental effects. Genetic variation is substantial and each individual of a species, with the exception of monozygotic twins, possesses a unique DNA sequence. DNA variations are mutations resulting from the substitution of single nucleotides (single nucleotide polymorphisms—SNPs), the insertion or deletion of DNA fragments of various lengths (from a single to several thousand nucleotides), or the duplication or inversion of DNA fragments (Marsjan and Oldenbroek 2007). For this reason, the genome is composed of four different nucleotides (A, C, T, and G). Next, we provide two important definitions that are keys to understanding how markers are used in genomic selection.

Genetic markers A genetic marker is a gene or DNA sequence with a known location on a chromosome and is generally used to identify individuals, which is why it is a powerful tool to explore genetic diversity. It can be described as a variation that may arise due to a mutation or alteration in the genomic loci that can be observed. A genetic marker may be a short DNA sequence, such as a sequence surrounding a single base-pair change (single nucleotide polymorphism, SNP, see Fig. 2.1), or a long one, such as mini- and microsatellites. Molecular markers can be used in molecular biology and biotechnology to identify a particular DNA sequence in a pool of unknown DNA. For example, DNA is used to search for useful genes, and also for marker-assisted selection, paternity testing, and food traceability. In GS, genetic markers measured across the genome are used to measure genomic similarities between individuals; in theory, this can be more precise than pedigree information.

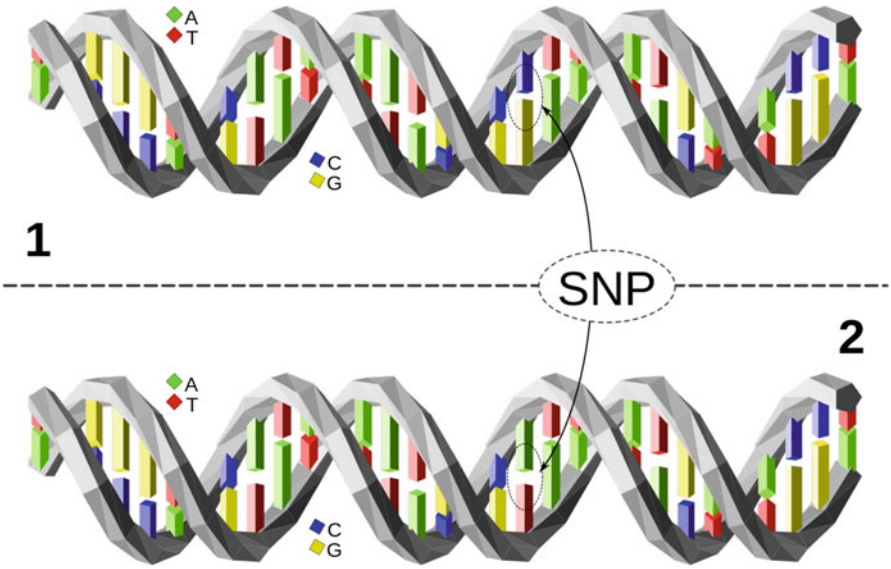

Fig. 2.1 The upper DNA molecule differs from the lower DNA molecule at a single base-pair location (a C/A polymorphism)

Markers can be used to estimate the proportion of chromosome segments shared by individuals, including the identification of genes that are identical by state (IBS). It is important to point out that probabilities generated from pedigree (A matrix) are discrete between close relatives. For example, full sibs share 0.5 of alleles (genome) that are identical by descent, that is, that are inherited from a common ancestor. A simple nucleotide polymorphism (SNP) is a widely used marker.

As an initial descriptive analysis of this type of data that could help delve into some characteristics of the structure of the genotypes data, we can compute the frequency genotypes in our data, basically the proportion of each genotype, and can help us as description of the gene variation. Rather, a more common description of the genetic variation is the allele frequency, which is the proportion of each allele present in the population, and when the individual is a diploid, its contribution of the proportion is of two alleles for each gene (Griffiths et al. 2005). In a diploid population with two alleles, A and a, if n_{A}, n_{Aa}, and n_{a} are the frequencies of the three genotypes (AA, Aa, aa) in a sample of n individuals, the frequency of alleles A and a are given by $\frac{2n_{\mathrm{A}}+n_{\mathrm{Aa}}}{2n} = \frac{n_{\mathrm{A}}}{n} + \frac{n_{\mathrm{Aa}}}{2n}$ and $\frac{2n_{\mathrm{a}}+n_{\mathrm{Aa}}}{2n} = \frac{n_{\mathrm{a}}}{n} + \frac{n_{\mathrm{Aa}}}{2n}$, respectively. In this case, the allele with the lower frequency is called the minor allele and the other, the major allele.

To illustrate marker recoding and depuration, consider the following example of eight plants genotyped for seven SNPs.

With the information in Table 2.5, we first proceed to find the minor (less common) and major (most common) alleles of each marker. In marker 1 (SNP1),

Table 2.5 Marker information for eight plants and seven SNPs denoted as SNP1, . . ., SNP7

Plant	SNP1	SNP2	SNP3	SNP4	SNP5	SNP6	SNP7
1	C_G	C_C	T_T	G_T	G_G	T_C	G_G
2	C_G	C_G	A_A	G_T	C_C	C_C	G_C
3	C_C	?_?	T_A	G_T	G_C	T_C	?_?
4	G_G	?_?	T_T	T_T	G_C	T_C	G_C
5	C_C	C_G	T_A	G_T	G_C	T_T	C_C
6	?_?	C_G	T_A	G_T	C_C	?_?	G_G
7	C_G	C_C	T_A	G_G	G_C	C_C	G_C
8	C_G	C_C	T_A	G_G	G_C	T_C	G_G

?_? denotes a missing genotype

Table 2.6 Minor allele (minorAllele) and major allele (majorAllele) for each marker

Marker	minorAllele	majorAllele
SNP1	G	C
SNP2	G	C
SNP3	A	T
SNP4	T	G
SNP5	G	C
SNP6	T	C
SNP7	C	G

we can see that the minor allele is G, since it only appears in G_G in one out of the eight plants; the major allele is C, since C_C appears in two out of eight plants. In SNP2, the minor allele is G since it does not appear in any of the eight plants, whereas C_C appears in three out of eight plants. Due to this, C is the major allele. In SNP3, the minor and major alleles are A (with A_A observed in 1/8) and T (with T_T observed in 2/8), respectively. Using this logic, Table 2.6 shows the minor and major alleles of each of the markers.

Once we have this information (minor and major alleles), it can be used to fit additive effects models, but it is first necessary to recode the marker information following the rules below:

$$x = \begin{cases} 0 \text{ if the SNP is homozygous for the major allele} \\ 1 \text{ if the SNP is heterozygous} \\ 2 \text{ if the SNP is homozygous for the other allele} \end{cases}$$

The recoded information in terms of additive effects is given in Table 2.7, where the recording is now in terms of 0, 1, and 2, following the above rules. In turn, the missing genotypes are recoded as NA.

Next, we show the minor allele frequency (MAF), the frequency of NAs and the frequency of heterozygotes genotypes (freHetero) for each marker. The frequency allele can be computed from the frequency of genotypes as described earlier and from there, the minor allele and its frequency can be deduced, but once the marker

Table 2.7 Marker information recoded as 0, 1, and 2, for eight plants and seven SNPs denoted as SNP1, . . ., SNP7

Plant	SNP1	SNP2	SNP3	SNP4	SNP5	SNP6	SNP7
1	1	0	0	1	2	1	0
2	1	1	2	1	0	0	1
3	0	NA	1	1	1	1	NA
4	2	NA	0	2	1	1	1
5	0	1	1	1	1	2	2
6	NA	1	1	1	0	NA	0
7	1	0	1	0	1	0	1
8	1	0	1	0	1	1	0

NA denotes a missing genotype

Table 2.8 Minor allele frequency (MAF), frequency of NAs (freqNA), and frequency of heterogeneous (freqHetero) are reported for each marker

Marker	minorAllele	majorAllele	MAF	freqNA	freqHetero
SNP1	G	C	0.429	0.125	0.571
SNP2	G	C	0.250	0.250	0.500
SNP3	A	T	0.438	0.000	0.625
SNP4	G	T	0.438	0.000	0.625
SNP5	G	C	0.438	0.000	0.625
SNP6	T	C	0.429	0.125	0.571
SNP7	C	G	0.357	0.125	0.429

information is coded as Table 2.7, in an equivalent way, the MAF also can be calculated as the mean of each column in Table 2.7 divided by 2 without taking into account the missing values, that is, as MAF $= \left(\sum_{i=1}^{n_c} x_i^*\right)/(2n_c)$, where $x_i^*, i = 1, \ldots, n_c$ are the coded genotyped values for non-missing individual values for a marker. For example, the MAF of marker SNP2 is equal to $\text{MAF}_{\text{SNP2}} = \frac{0+1+1+1+0+0}{2(6)} = \frac{3}{12} = 0.25$. See Table 2.8 for the MAF of the remaining markers, where the frequency of NAs (freqNA) is also reported for each marker, which corresponds to the number of missing values (NAs) in each column divided by the number of individuals (8). Furthermore, the frequency of heterogeneous genotypes (freqHetero) was calculated (ratio of the summation of only the ones divided by the non-missing values).

With the data formatted in this way, we are ready to compute a genomic relationship matrix or matrix of realized genetic similarities among all pairs of individuals.

It is important to point out that the recoding process for values of 0, 1, and 2 can be performed automatically using the library synbreed, but we first need to load the complete information onto R of Table 2.5. Then, for recording and imputing, we used methods available from this library; the code used for these tasks is given in Appendix 1. The output is explained below.

First, we called the library synbreed and then we loaded the marker information contained in a file called MarkersToy.csv, which is saved in an object called snp7. When all the marker information is printed, we can clearly see that it corresponds to the information given in Table 2.5, although without the first column.

```
> library(synbreed)
> ####Loading the marker information
> snp7 <- read.csv("MarkersToy.csv",header=T)
> snp7=snp7[,-1]
> snp7
  SNP1 SNP2 SNP3 SNP4 SNP5 SNP6 SNP7
1  C_G  C_C  T_T  G_T  G_G  T_C  G_G
2  C_G  C_G  A_A  G_T  C_C  C_C  G_C
3  C_C  ?_?  T_A  G_T  G_C  T_C  ?_?
4  G_G  ?_?  T_T  T_T  G_C  T_C  G_C
5  C_C  C_G  T_A  G_T  G_C  T_T  C_C
6  ?_?  C_G  T_A  G_T  C_C  ?_?  G_G
7  C_G  C_C  T_A  G_G  G_C  C_C  G_C
8  C_G  C_C  T_A  G_G  G_C  T_C  G_G
```

Next, we rename the rows of the object snp7 (matrix of marker information) coded with values of ID1 to ID8. We then select the position in this matrix of values equal to ?_?, followed by these values being replaced with NA. Finally, the snp7 object is printed again and we can see that the values of ?_? were replaced by NAs.

```
> ####Set names for individuals
> rownames(snp7) <- paste("ID",1:8,sep="")
> pos.NA=which(snp7=="?_?", arr.ind=TRUE)
> snp7[pos.NA]=NA
> snp7
    SNP1 SNP2  SNP3  SNP4 SNP5 SNP6 SNP7
ID1  C_G  C_C  T_T  G_T  G_G  T_C  G_G
ID2  C_G  C_G  A_A  G_T  C_C  C_C  G_C
ID3  C_C  NA   T_A  G_T  G_C  T_C  NA
ID4  G_G  NA   T_T  T_T  G_C  T_C  G_C
ID5  C_C  C_G  T_A  G_T  G_C  T_T  C_C
ID6  NA   C_G  T_A  G_T  C_C  NA   G_G
ID7  C_G  C_C  T_A  G_G  G_C  C_C  G_C
ID8  C_G  C_C  T_A  G_G  G_C  T_C  G_G
```

Later, the object with the marker information, snp7, is transformed into an object of class gpData.

```
> ####Creating an object of class 'gpData'
> gp <- create.gpData(geno=snp7)
```

Using the function codeGeno() and giving the marker object, gp, as an input, we recode the marker information to values of 0, 1, and 2. The recoded marker information is then extracted and printed, and here we can see that the recoding

performed with this library is exactly equal to what we obtained manually and presented in Table 2.7.

```
> ####Recoding to 0, 1, and 2 values the genotypic data
> gp.coded <-codeGeno(gp)
> Geno_Recoded=gp.coded$geno
> Geno_Recoded
      SNP1 SNP2 SNP3 SNP4 SNP5  SNP6 SNP7
ID1   1     0     0     1     2     1     0
ID2   1     1     2     1     0     0     1
ID3   0     NA    1     1     1     1     NA
ID4   2     NA    0     2     1     1     1
ID5   0     1     1     1     1     2     2
ID6   NA    1     1     1     0     NA    0
ID7   1     0     1     0     1     0     1
ID8   1     0     1     0     1     1     0
```

Finally, we once again use the function codeGeno(), but we also add input=T and inpute.type="random", which will recode the object gp to values 0, 1, and 2, as done previously. However, it will also input the missing cells with NAs using the random method of imputation. Finally, the matrix of markers coded with values 0, 1, and 2 is presented, but with the NAs inputted with values of 0, 1, and 2 using the random method. It is important to point out that this library has other imputation options such as "family," "beagle," "beagleAfterFamily," "beagleNoRand," "beagleAfterFamilyNoRand," and "fix," but the technical details of each imputation method go beyond the scope of this book.

```
> ####Recoding to values of 0, 1, and 2 the genotypic data and inputting
> Imputed_Geno<-codeGeno(gp,impute=T,impute.type="random")
> Imputed_Geno$geno
     SNP1 SNP2 SNP3 SNP4 SNP5 SNP6 SNP7
ID1   1    0    0    1    2    1    0
ID2   1    1    2    1    0    0    1
ID3   0    0    1    1    1    1    2
ID4   2    0    0    2    1    1    1
ID5   0    1    1    1    1    2    2
ID6   2    1    1    1    0    0    0
ID7   1    0    1    0    1    0    1
ID8   1    0    1    0    1    1    0
```

2.4 Methods to Compute the Genomic Relationship Matrix

The three methods described here to calculate the genomic relationship matrix (GRM) are based on VanRaden's (2008) paper "Efficient methods to compute genomic predictions" where more theoretical support for each of these methods can be found. We assume that we have a matrix of markers of order $J \times p$, where J denotes the number of lines and p the number of markers, and that this matrix does

not contain missing values and is coded as 0, 1, and 2, or −1, 0, and 1 to refer homozygotes major allele, heterozygous, and homozygous minor allele, respectively. Note that the last codification is related to the first by the relation $\boldsymbol{X}_2 = \boldsymbol{X} + \mathbf{1}_J \mathbf{1}_p^{\mathrm{T}}$, where $\boldsymbol{X}_2$ is a matrix of markers information coded in terms of −1, 0, and 1, while $\boldsymbol{X}$ is the coded marker information in terms of 0, 1, and 2, and $\mathbf{1}_q$ is the column vector of dimension q with ones in all its entries.

Method 1. This method calculates the GRM as

$$\boldsymbol{G} = \frac{1}{p}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}},$$

where $\boldsymbol{X}$ is the matrix of marker genotypes of dimensions $J \times p$. When the marker information is coded as −1, 0, and 1 as described before, the diagonal terms of $p\boldsymbol{G}$ count the number of homozygous loci for each line, and the off-diagonal of $p\boldsymbol{G}$ is a measure of the number of alleles shared by two lines (VanRaden 2008). To illustrate how to calculate the GRM under this method, we will use the matrix of marker genotypes obtained in the previous section with Imputed_Geno$geno, which in the matrix format is equal to

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 & 0 & 1 & 2 & 1 & 0 \\ 1 & 1 & 2 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 2 \\ 2 & 0 & 0 & 2 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Then, using the R code, we calculate the GRM under this first method as

```
##Computing the genomic relationship matrix—Method1
> G_M1=tcrossprod(X)/dim(X)[2]
> G_M1
     ID1   ID2   ID3   ID4   ID5   ID6   ID7   ID8
ID1 0.875 0.250 0.500 0.875 0.625 0.375 0.375 0.500
ID2 0.250 1.000 0.625 0.625 0.750 0.750 0.500 0.375
ID3 0.500 0.625 1.000 0.750 1.125 0.250 0.500 0.375
ID4 0.875 0.625 0.750 1.375 0.875 0.750 0.500 0.500
ID5 0.625 0.750 1.125 0.875 1.500 0.375 0.500 0.500
ID6 0.375 0.750 0.250 0.750 0.375 0.875 0.375 0.375
ID7 0.375 0.500 0.500 0.500 0.500 0.375 0.500 0.375
ID8 0.500 0.375 0.375 0.500 0.500 0.375 0.375 0.500
```

Method 2. In this method the GRM is similar to method 1, but first each marker is centered by twice the minor allele frequency:

$$\boldsymbol{G} = \frac{(\boldsymbol{X} - \boldsymbol{\mu_E})(\boldsymbol{X} - \boldsymbol{\mu_E})^{\mathrm{T}}}{2\sum_{j=1}^{p} p_j(1 - p_j)},$$

where p_j is the minor allele frequency (MAF) of SNP $j = 1, \ldots, p$ and $\boldsymbol{\mu_E}$ is the expected value of matrix $\boldsymbol{X}$ under the Hardy–Weinberg equilibrium (Griffiths et al. 2005) from estimates of allelic frequencies, that is, $\boldsymbol{\mu_E} = \mathbf{1}_{\boldsymbol{J}}[2p_1, \ldots, 2p_p]$. Term $2\sum_{j=1}^{p} p_j(1 - p_j)$ is the sum of the variance estimates of each marker and makes GRM analogous to the numerator relationship matrix (VanRaden 2008).

Now, using the following R code, we calculate the GRM under method 2 as

```
##Computing the genomic relationship matrix—Method2
> phat=colMeans(X)/2#Minor allele frequency
> phat
 SNP1   SNP2   SNP3   SNP4   SNP5   SNP6   SNP7
 0.5000 0.1875 0.4375 0.4375 0.4375 0.3750 0.4375

> X2=scale(X,center=TRUE,scale=FALSE)
> k=2*sum(phat*(1-phat))
> G_M2=tcrossprod(X2)/k

> round(G_M2,3)
     ID1    ID2    ID3    ID4    ID5    ID6    ID7    ID8
ID1  0.930 -0.766 -0.227  0.352 -0.265 -0.227 -0.072  0.275
ID2 -0.766  0.930 -0.072 -0.419 -0.111  0.545  0.082 -0.188
ID3 -0.227 -0.072  0.776 -0.188  0.737 -0.766  0.005 -0.265
ID4  0.352 -0.419 -0.188  1.007 -0.227  0.120 -0.342 -0.304
ID5 -0.265 -0.111  0.737 -0.227  1.316 -0.805 -0.342 -0.304
ID6 -0.227  0.545 -0.766  0.120 -0.805  1.084  0.005  0.043
ID7 -0.072  0.082  0.005 -0.342 -0.342  0.005  0.467  0.198
ID8  0.275 -0.188 -0.265 -0.304 -0.304  0.043  0.198  0.545
```

Method 3. Under this method, the GRM should be calculated as

$$\boldsymbol{G} = \frac{\boldsymbol{Z}\boldsymbol{Z}^{\mathrm{T}}}{p},$$

where $\boldsymbol{Z}$ is the matrix of scaled SNP codes and p is the number of SNPs, that is $z_{ij} = (x_{ij} - 2p_j)/\sqrt{2p_j(1 - p_j)}$.

Finally, for this third method, the GRM can be calculated as

```
> ##Computing the genomic relationship matrix—Method3
> X3=scale(X,center=TRUE,scale=TRUE)
> G_M3=tcrossprod(X3)/ncol(X3)
> round(G_M3,3)
     ID1    ID2    ID3    ID4    ID5    ID6    ID7    ID8
ID1  0.962 -0.880 -0.093  0.435 -0.221 -0.397 -0.028  0.223
ID2 -0.880  1.084 -0.133 -0.507 -0.014  0.667  0.012 -0.228
ID3 -0.093 -0.133  0.619 -0.112  0.490 -0.658  0.023 -0.136
ID4  0.435 -0.507 -0.112  1.058 -0.241  0.022 -0.350 -0.305
ID5 -0.221 -0.014  0.490 -0.241  1.181 -0.539 -0.391 -0.265
ID6 -0.397  0.667 -0.658  0.022 -0.539  1.053 -0.057 -0.092
ID7 -0.028  0.012  0.023 -0.350 -0.391 -0.057  0.516  0.276
ID8  0.223 -0.228 -0.136 -0.305 -0.265 -0.092  0.276  0.527
```

2.5 Genomic Breeding Values and Their Estimation

In plant and animal breeding, it is a common practice to rank and select individuals (plants or animals) based on their true breeding values (TBVs), also called additive genetic values. However, since we cannot see genes and breeding values, this task is not straightforward, and it is therefore estimated indirectly using observed phenotypes. The estimated values are called estimated breeding values (EBVs), which means that TBV is a latent variable that is only approximated using the observable variable (phenotype).

When the TBVs are used, the genetic change is expected to be larger than when the EBVs are used, but this difference is small when the EBVs are accurately estimated. EBVs reflect the true genetic potential or true genetic transmitting ability of individuals (plants or animals). Traditionally, they are estimated based on the performance records of their parents, sibs, progenies, and their own after correcting for various environmental factors such as management, season, age, etc. When parents are selected based on their breeding values with high reliability, a faster genetic progress is expected in the resulting population. For this reason, the process of estimating breeding values is of paramount importance in any breeding program.

There are several methods to estimate genomic estimated breeding values (GEBVs), but first we will describe the best linear unbiased predictor (BLUP) method. When using the BLUP method to estimate the GEBVs, we need to use the mixed model equations (2.2) described above to estimate BLUEs and BLUPs. Using this equation (2.2) but depending on the form taken by the matrices $\boldsymbol{Z}$ and $\boldsymbol{\Sigma}$, we can end up with the GBLUP method or the SNP-BLUP method to estimate the breeding values. First, we explain the GBLUP method, where we substitute $\boldsymbol{Z}$ and $\boldsymbol{\Sigma}$ matrices for the incidence matrix of genotypes and genomic relationship matrix (GRM) derived from allele frequencies calculated with one of the methods of

VanRaden (2008) given in Sect. 2.4. Under this GBLUP method, the GEBV can be obtained as the solution $\widehat{\boldsymbol{u}}$ of the mixed model equation:

$$\begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\boldsymbol{u}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\mathbf{1} \\ \mathbf{1}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \mathbf{1}^{\mathrm{T}}\boldsymbol{R}^{-1}\mathbf{1} + \sigma_g^{-2}\,\boldsymbol{G}^{-1} \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \\ \mathbf{1}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \end{pmatrix}, \tag{2.3}$$

where $\boldsymbol{Z}$ was replaced by $\boldsymbol{Z} = \mathbf{1}$ and $\boldsymbol{\Sigma}$ by $\sigma_g^2\boldsymbol{G}$, the genomic relationship matrix that was calculated with some of the methods described in Sect. 2.4 and the genomic variance component (σ_g^2) should be estimated. We ended up with the system of equations given in Eq. (2.3), since the model used is $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{u} + \boldsymbol{\varepsilon}$, with $\boldsymbol{u} \sim N\left(\mathbf{0}, \sigma_g^2\boldsymbol{G}\right)$, where σ_g^2 is the genomic variance component, $\boldsymbol{G}$ is a GRM of dimension $q \times q$ calculated using any of the three methods given in Sect. 2.4, and the other terms are exactly as in Eq. (2.2). One of the greatest advantages of using the GBLUP method to obtain GEBV is that the dimensionality of the design matrices is, at most, equal to the number of lines under study. On the other hand, under the SNP-BLUP, we substitute the $\boldsymbol{Z}$ and $\boldsymbol{\Sigma}$ matrices in Eq. (2.2) with $\boldsymbol{M}$ (scaled marker information matrix of order $n \times p$) and $\sigma_M^2\boldsymbol{I}$, respectively. Under this SNP-BLUP method, the mixed model equation is equal to

$$\begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\boldsymbol{u}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{M} \\ \boldsymbol{M}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{M}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{M} + \sigma_M^{-2}\boldsymbol{I}^{-1} \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \\ \boldsymbol{M}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \end{pmatrix} \tag{2.4}$$

Under this mixed model equation, $\boldsymbol{u}$ is now the random effects of markers, and therefore, to obtain the GEBV, we use the estimates of marker effects ($\widehat{\boldsymbol{u}}$) and GEBV = $\boldsymbol{M}\widehat{\boldsymbol{u}}$, since now the model used is $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{Mu} + \boldsymbol{\varepsilon}$, with $\boldsymbol{u} \sim N\left(\mathbf{0}, \sigma_M^2\boldsymbol{I}\right)$.

We then illustrate how to estimate the GEBV under both BLUP methods. For this purpose, we provide, in Table 2.9, a data set with eight lines (evaluated in two environments) for grain yield, for which, in turn, we use the seven markers imputed in Sect. 2.3.

Below are the data of Table 2.9 that were saved in the data.for.GEBV.csv file. Library rrBLUP was used to estimate the GEBV using the mixed model equations.

```
> library(rrBLUP)
> data=read.csv("data.for.GEBV.csv")
> data
  X Env Lines y      SNP1 SNP2 SNP3 SNP4 SNP5 SNP6 SNP7
1 1 E1  L1    5.215  1    0    0    1    2    1    0
2 2 E1  L2    4.998  1    1    2    1    0    0    1
3 3 E1  L3    5.284  0    0    1    1    1    1    2
4 4 E1  L4    5.157  2    0    0    2    1    1    1
5 5 E2  L5    6.601  0    1    1    1    1    2    2
6 6 E2  L6    5.735  2    1    1    1    0    0    0
7 7 E2  L7    5.565  1    0    1    0    1    0    1
8 8 E2  L8    5.829  1    0    1    0    1    1    0
```

Table 2.9 Grain yield (y) of eight lines in two environments

Env	Lines	y	SNP1	SNP2	SNP3	SNP4	SNP5	SNP6	SNP7
E1	L1	5.215	1	0	0	1	2	1	0
E1	L2	4.998	1	1	2	1	0	0	1
E1	L3	5.284	0	0	1	1	1	1	2
E1	L4	5.157	2	0	0	2	1	1	1
E2	L5	6.601	0	1	1	1	1	2	2
E2	L6	5.735	2	1	1	1	0	0	0
E2	L7	5.565	1	0	1	0	1	0	1
E2	L8	5.829	1	0	1	0	1	1	0

In matrix M, only the columns corresponding to marker information are selected. This information is scaled by column using the scale command of R, and the scaled markers are saved in the MS matrix; the GRM is calculated with this information using method 3 in Sect. 2.4. Here we obtained the design matrix for environments and lines, and we also added the genomic information to the design matrix of lines by post-multiplying the design matrix of lines by the Cholesky decomposition of the GRM, which also can be used as an alternative way to obtain the breeding values. This is because the GBLUP model (2.1) can be expressed equivalently as follows:

$$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{Z}^{*}\boldsymbol{u}^{*} + \boldsymbol{\varepsilon},$$

where now $\boldsymbol{Z}^{*} = \boldsymbol{ZL}^{\mathrm{T}}$, $\boldsymbol{G} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ is the Cholesky decomposition of $\boldsymbol{G}$, and $\boldsymbol{u}^{*}$ is a random vector with distribution $N\left(\boldsymbol{0}, \sigma_g^2 \boldsymbol{I_q}\right)$.

```
> M=data[,5:11]
> MS=scale(M) #Scales matrix of markers
> G=MS%*%t(MS)/ncol(MS) #Genomic relationship matrix method 3
> X_E=model.matrix(~0+Env,data=data) #Matrix design of environments
> X_L1=model.matrix(~0+Lines,data=data)#Matrix design of lines
> L=chol(G) #Cholesky decomposition of G
> X_L=X_L1%*%t(L)#Modified matrix design of lines, Z*
```

Then, using the mixed.solve() function of the rrBLUP package, providing as input the response variable (y), the X_L1 matrix of lines and the GRM matrix, $\boldsymbol{G}$, we solved the mixed model equation (2.3) and obtained the GEBVs, which are extracted using fm1$u.

```
> ########################Solution GBLUP#############################
> y = data$y
> fm1=mixed.solve(y=y, Z=X_L1, K=G, X=X_E, method="REML",
+       bounds=c(1e-09, 1e+09), SE=FALSE, return.Hinv=FALSE)
> fm1$u
[1] 0.08371533 -0.13384553 0.15177641 0.02540245 0.63649420
-0.22942366 -0.39877555 -0.13534366
####################Alternative solution GBLUP######################
```

```
> fm1a=mixed.solve(y=y, Z=X_L, K=diag(dim(G)[1]), X=X_E,
method="REML",
        bounds=c(1e-09, 1e+09), SE=FALSE, return.Hinv=FALSE)
>#GEBV
>X_L%*%fm1a$u
```

Shown below is the code to obtain the GEBV, but using the SNP-BLUP method and also using the mixed.solve() function. However, instead of giving the GRM as input, this is given by $\boldsymbol{Z}$, which is the MS matrix, containing the matrix of the markers scaled. Now the random effects of markers are extracted with fm2$u, and to obtain the GEBV, these random effects are pre-multiplied by the scaled design matrix of markers:

```
> ########################Solution SNP-BLUP#########################
> fm2=mixed.solve(y=y, Z=MS, X=X_E, method="REML",
+       bounds=c(1e-09, 1e+09), SE=FALSE, return.Hinv=FALSE)
> fm2$u
SNP1          SNP2         SNP3         SNP4         SNP5         SNP6         SNP7
-0.121863421 0.115618300 -0.040677892 0.083526359 0.016201822 0.184761140 -0.001827558
> beta_Mar=fm2$u
> GEBV=c(MS%*%beta_Mar)
> GEBV
[1] 0.08373722 -0.13385647 0.15181358 0.02538892 0.63650107 -0.22940375 -0.39883111
-0.13534946
```

This shows that both methods (GBLUP and SNP-BLUP), give exactly the same breeding value estimates. However, although the two methods are specified in almost the same way and give similar estimates, which, based on all the SNPs (SNP-BLUP), required more computational time, this difference will be more notable for larger data sets that containing larger number of markers. Due to this, in these situations we can choose the GBLUP method.

Some advantages of using the Henderson equation to obtain the GEBV are

(a) It fits nicely into existing BLUP software and into existing theory.
(b) It provides measures of accuracy from the inverse of the LHS (Linear Henderson system).
(c) It accommodates all individuals (plants or animals).

However, it has some inconveniences, such as

(a) It can't easily accommodate major genes (unless using weights in the construction of $\boldsymbol{G}$).
(b) Computation of $\boldsymbol{G}$ and inversion might be challenging.

It is important to point out that the GEBV can be obtained using other estimation methods such as Bayesian methods. Bayesian methods for GS are explained in detail in upcoming chapters, but here we will illustrate the use of the BGLR package to estimate the GBLUP Bayesian method and the SNP-BLUP Bayesian method.

We first provide the code for the GBLUP Bayesian method. The predictor ETA is a list and has two sub-list components: the first, for the effects of environments for which a FIXED model (model="FIXED") is used that uses a prior non-informative for each beta coefficient. The second component is an RKHS model to specify the distribution of the random effects of lines that, in general, are of the form $N\left(\mathbf{0}, \sigma_g^2 \boldsymbol{K}\right)$, and in this case, using the GRM, $\boldsymbol{K} = \boldsymbol{G}$. Then the Bayesian GBLUP model is fitted using the function BGLR(), and finally, the GEBVs are obtained with fm1ETAGen$u. The response vector value is specified in the first option of the function BGLR, y=y, and with nIter=20000 and burnIn=10000, the number of iterations to run the involved MCMC method are specified, along with the number of these that will be discarded at the beginning of the MCMC, respectively.

```
> library(BGLR)
> ######################GBLUP-BLUP Bayesian#########################
> ETA=list(Env=list(X=X_E[,-1], model="FIXED"),Gen=list(K=G,
model="RKHS"))
> fm1=BGLR(y=y,ETA=ETA,nIter=20000,burnIn=10000,verbose=F)
> fm1$ETA$Gen$u
[1] 0.02581315 -0.08430838 0.12806091 0.03949064 0.32406895 -0.17367228
[7] -0.16452623 -0.09492676
```

The SNP-BLUP Bayesian GEBV is then fitted, but now also providing the design matrix of environments as input, and the scaled design matrix of markers in the second term. The model now used is a Bayesian Ridge regression (BRR) that gives a normal distribution with mean zero and a common variance component as a prior for each marker effect.

```
> ########################SNP-BLUP Bayesian#########################
> ETA=list(Env=list(X=X_E[,-1], model="FIXED"), Gen=list(X=MS,
model="BRR"))
> fm1=BGLR(y=y,ETA=ETA,nIter=20000,burnIn=10000,verbose=F)
> beta_Mar_Bayes=fm1$ETA$Gen$b
>
> GEBV_Bayes=c(MS%*%beta_Mar_Bayes)
> GEBV_Bayes
[1] 0.03187469 -0.09663427 0.12920479 0.04973765 0.34695500
-0.18172760
[7] -0.18217036 -0.09723989
```

Here we can observe that both methods gave GEBVs that are very similar yet slightly different due to the Monte Carlo sampling. However, these Bayesian GEBVs are different from those obtained above using Henderson's mixed model equations, due to the fact that different machineries are used to estimate the GEBVs. Details of the Bayesian methods for GS will be provided in upcoming chapters.

Finally, it is important to point out that the advantage of the GBLUP over the SNP-BLUP is that the system of equations, when fitting the mixed model equations, is of the size of the individuals (lines or animals), which are, most of the time, fewer

than the number of markers (SNPs). This advantage is also observed under the Bayesian version, because the design matrix of markers is usually larger than the dimension of the genomic relationship matrix.

2.6 Normalization Methods

This section describes four types of normalization variables (inputs and outputs). In this case, normalization refers to the process of adjusting the different inputs or outputs that were originally measured in different scales to the same scale. It is very important to carry out the normalization process before giving the inputs and outputs for most statistical machine learning algorithms because it helps improve the numerical stability in the estimation process of some algorithms; it is suggested mostly when the inputs or outputs are in different scales. However, it is important to point out that in some statistical machine learning software, the normalization process is done internally, in which case this process does not need to be carried out manually. The five normalization methods we describe next are centering, scaling, standardization, max normalization, and minimax normalization.

Centering This normalization consists of subtracting from each variable (input or output) its mean, μ; this means that the centered values are calculated as

$$X_i^* = X_i - \mu$$

The centered variable X_i^* has a mean of zero.

Scaling This normalization consists of dividing each variable (input or output) by its standard deviation, σ. The scaled values are calculated as

$$X_i^* = \frac{X_i}{\sigma}.$$

The scaled variable X_i^* has unit variance.

Standardization This process of normalization consists of calculating its mean, μ, and standard deviation, σ, for each input or output. The standardized values are then calculated as

$$X_i^* = \frac{X_i - \mu}{\sigma}.$$

This process is carried out for each input or output variable, and this needs to be done with care, since we need to use the corresponding mean and standard deviation of each variable. The output of the standardized score has a mean of zero and a variance of one, which means that most standardized values range between -3.5 and 3.5.

Max normalization This normalization consists of dividing the values of the input or output by the maximum (max) value of this variable, meaning that this score is calculated as

$$X_i^* = \frac{X_i}{\max}$$

This normalization can be useful when there are no negative inputs, which guarantees that the normalized variable will be between 0 and 1.

Minimax normalization To implement this normalization, we first need to calculate the minimum (min) and maximum (max) value for each input or output; then the minimax score is calculated using the following expression:

$$X_i^* = \frac{X_i - \min}{\max - \min}$$

The resulting score of the minimax normalization is between 0 and 1. An inconvenience of this normalization method is that inputs or outputs with long-tail distributions will be dominated by inputs or outputs with uniform distributions.

It is important to point out that the normalization process is not limited to the independent variables (inputs), but can also be used for the dependent variables (outputs) when dealing with multiple outcomes in different scales. However, the normalization process of the dependent variables is not necessary for univariate prediction models or when developing a machine to predict mixed outcomes, because the original scale of the distributions can be losses, for example, to predict two types of outcomes (binary and continuous), to predict three types of outcomes (ordinal, continuous, and count), or to predict four types of outcomes (binary, ordinal, continuous, and count data). On the other hand, when developing a machine to predict four continuous outcomes in different scales (for example, grain yield in tons by hectare, plant height in centimeters, days to maturity in 0 to 120 days, and vitamin content in milligrams), in these cases, normalizing each of the response variables is suggested to avoid the training process from being dominated by the dependent variable with large variability, which implies that the trained machine would be able to predict only this response variable with the highest accuracy. However, in some statistical machine learning models, it is not necessary to normalize the dependent variables because they allow the user to put different weights on each dependent variable to be able to train the model more fairly.

2.7 General Suggestions for Removing or Adding Inputs

The following is a general guide to removing inputs:

(a) Remove an independent variable (input) if it has zero variance, which implies that the input has a single unique value (Kuhn and Johnson 2013).

(b) Remove an independent variable (input) if it has near-zero variance, which implies that the input has very few values.
(c) Remove an independent variable (input) if it is highly correlated with another input variable (nearly perfect correlation), since they are measuring the same underlying information (Kuhn and Johnson 2013). Known as collinearity in statistical machine learning science, this phenomenon is important because in its presence the parameter estimates of some machine learning algorithms (for example, those based on gradient descent) are inflated (not accurately estimated).

These three issues are very common in genomic prediction, since part of the independent variables is marker information and many of them have zero or near-zero variance and other pairs have very high correlations. One of the advantages of removing input information prior to the modeling process is that this reduces the computational resources needed to implement the statistical machine learning algorithm. Also, it is possible to end up with a more parsimonious and interpretable model. Another advantage is that models with less correlated inputs are less prone to unstable parameter estimates, numerical errors, and degraded prediction performance (Kuhn and Johnson 2013).

The following are general rules for the addition of input variables:

(a) Create dummy variables from nominal or categorical inputs.
(b) Manually create a categorical variable from a continuous variable.
(c) Transform the original input variable using a specific transformation.

First, we describe the process of creating dummy variables from categorical (nominal or ordinal) inputs. Transforming categorical inputs into dummy variables is required in most supervised statistical machine learning methods, since providing the original independent variable (not transformed into dummy variables) is incorrect and should be avoided by practitioners of statistical machine learning methods. However, it is important to point out that when the dependent variable is categorical, most statistical machine learning methods do not require it to be transformed into dummy variables. For example, assume that we are studying three genotypes (G1, G2, and G3) in two environments (E1 and E2) and we collected the following grain yield data.

Using the information in Table 2.10, we created the dummy variables for each categorical variable. First, we provide the dummy variables for the environments (Table 2.11).

Next, we provide the dummy variables for the genotypes (Table 2.12).

It is important to point out that in R we can use the model.matrix() to create dummy variables from categorical independent variables. First, we create a data frame called grain.yield with the original data set:

```
grain.yield=data.frame(Environment=c("E1","E1", "E1", "E2", "E2",
"E2"), Genotype=c("G1","G2","G3", "G1","G2","G3"), y=c(5.3,
5.6,5.8,6.5, 6.8,6.9))
```

Table 2.10 Grain yield was evaluated in two environments, and three genotypes were evaluated in each environment

Observation	Environment	Genotype	Grain yield (y)
1	E1	G1	5.3
2	E1	G2	5.6
3	E1	G3	5.8
4	E2	G1	6.5
5	E2	G2	6.8
6	E2	G3	6.9

Table 2.11 Resulting dummy variables for the environments

Observation	Env1	Env2
1	1	0
2	1	0
3	1	0
4	0	1
5	0	1
6	0	1

Table 2.12 Resulting dummy variables for genotypes

Observation	G1	G2	G3
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	1	0
6	0	0	1

We then print the data

```
> grain.yield
      Environment Genotype  y
  1    E1          G1        5.3
  2    E1          G2        5.6
  3    E1          G3        5.8
  4    E2          G1        6.5
  5    E2          G2        6.8
  6    E2          G3        6.9
```

Next, we create the dummy variables for the categorical variable environment using the model.matrix() function, as

```
ZE=model.matrix(~0+Environment, data=grain.yield)
```

The resulting matrix with the dummy variables of environments is called design matrix of environments and, in this case, it is

```
> ZE
   EnvironmentE1 EnvironmentE2
1              1             0
2              1             0
3              1             0
4              0             1
5              0             1
6              0             1
```

It is important to point out that, if instead of ~0+Environment, we use ~1-+Environment inside the model.matrix() function, we obtain a different form of the design matrix for environments:

```
> ZE
 (Intercept) EnvironmentE2
1           1             0
2           1             0
3           1             0
4           1             1
5           1             1
6           1             1
```

Strictly speaking, both design matrices contain the same information due to the fact that only $C - 1$ dummy variables are needed to capture all the information of the categorical variable, since once we have the information of $C - 1$ dummy variables, we can infer the information of the missing dummy variable. However, the decision to include all dummy variables depends on the selected statistical machine learning algorithm. Under the second version of the design matrix created with the model.matrix() function, in ZE, the column (dummy variable) corresponding to the first environment was not included, but instead an intercept (a column of ones in all rows) was added. This design matrix with an intercept is very important in some statistical machine learning algorithms such as most generalized regression models, neural networks, and deep learning models. In some cases, when this intercept is not included, numerical problems occur in the estimation of the learnable parameters (beta coefficients or weights, intercepts, etc.). The reason is that for each row (observation), these variables all add up to one, and this would provide the same information as the intercept (Kuhn and Johnson 2013). However, when statistical machine learning is not sensitive to not including the intercept (as the first design matrix), using the complete set of dummy variables can help improve model interpretation.

In the same way, the design matrix (dummy variables) for genotype is created using the following R code:

```
ZG=model.matrix(~0+Genotype, data=grain.yield)
```

which provides the following dummy variables that are accommodated in the ZG matrix:

```
> ZG
  GenotypeG1 GenotypeG2 GenotypeG3
1          1          0          0
2          0          1          0
3          0          0          1
4          1          0          0
5          0          1          0
6          0          0          1
```

This design matrix is composed of three columns because there are three categories for the categorical variable (G1, G2, and G3). Each column represents a genotype and a dummy variable was created for each genotype using an indicator variable of 0 (not present in that row) and 1 (present in that row) for each genotype. Now, using model.matrix(~1+Genotype, data=grain.yield), we obtain the design matrix with an intercept, but without the dummy variable for genotype 1:

```
> ZG
  (Intercept) GenotypeG2 GenotypeG3
1           1          0          0
2           1          1          0
3           1          0          1
4           1          0          0
5           1          1          0
6           1          0          1
```

As mentioned earlier, both design matrices for genotypes are valid since they contain the same information, given that in order to capture all the information of a categorical variable, reporting only $C - 1$ dummy variables in the design matrix is enough. However, the choice of one or another depends mostly on the statistical machine learning model to be used. It is also important to point out that the model.matrix() function, when containing the intercept, deletes the dummy variable corresponding to the first level of the categorical variable, but from the statistical point of view, any other dummy variable can be deleted without a loss of information if, and only if, $C - 1$ dummy variables are maintained.

Regarding the second point (b: manually create a categorical variable from a continuous variable), we refer to the process of manually converting a continuous variable to a categorical variable. For example, let us assume that we measured plant height in centimeters and then decided to categorize this response variable into five groups (group 1 if plant height is less than 100 cm, group 2 if plant height is between 100 and 125 cm, group 3 if it is between 125 and 150 cm, group 4 if it is between 150 and 175 cm, and group 5 when plant height is greater than 175 cm). This type of categorization is sometimes required, although we suggest avoiding categorizing continuous outcomes in this way, since a significant amount of information is lost and will affect the prediction performance of the trained model. In addition, it is

important to point out that the smaller the number of categories created, the greater the loss of information. Also, some researchers like Austin and Brunner (2004) reported that categorizing continuous inputs increases the rate of false positives (Kuhn and Johnson 2013). However, if researchers can justify that categorization is necessary, they should categorize the continuous input or output using a model framework to be able to do this with more precision.

Regarding the third approach to adding or creating inputs by transforming the original input variable using a specific transformation, we refer to the use of kernels, in which the input variables are transformed in such a way that the transformed input is used in the modeling process and the type of transformation depends on the objective of the study. This type of transformation with kernels will be discussed in detail in upcoming chapters. Also, many times transformations are applied to guarantee normality or any other distributional assumption on the variable of interest.

2.8 Principal Component Analysis as a Compression Method

Principal component analysis (PCA) is a method often used to compress the input data without losing as much information. The PCA works on a rectangular matrix in which the rows represent the observations (n) and the columns, the independent variables (p). The PCA creates linear combinations of the columns of matrix information, $\boldsymbol{X}$, and generates, at most, p linear combinations, called principal components. These linear combinations, or principal components, can be obtained as follows:

$$\begin{aligned} \text{PC}_1 &= \boldsymbol{w}_1\boldsymbol{X} = w_{11}X_1 + w_{12}X_2 + \cdots + w_{1p}X_p \\ &\cdots \\ \text{PC}_p &= \boldsymbol{w}_p\boldsymbol{X} = w_{p1}X_1 + w_{p2}X_2 + \cdots + w_{pp}X_p \end{aligned}$$

These linear combinations are constructed in such a way that the first principal component, PC_1, captures the largest variance, the second principal component, PC_2, captures the second largest variance, and so on. For this reason, it is expected that few principal components ($k < p$) can explain the largest variability contained in the original rectangular matrix ($\boldsymbol{X}$), which means that with a compressed matrix, $\boldsymbol{X}^*$, we contain most of the variability of the original matrix, but with a significant reduction in the number of columns. In matrix notation, the full principal components are obtained with the following expression:

$$\mathbf{PC} = \boldsymbol{XW},$$

where $\boldsymbol{W}$ is a p-by-p matrix of weights whose columns are the eigenvectors of $\boldsymbol{Q} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, that is, we first need to calculate the eigenvalue decomposition of $\boldsymbol{Q}$, which is equal to $\boldsymbol{Q} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^{\mathrm{T}}$, where $\boldsymbol{W}$ represents the matrix of eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix of order p-by-p containing the eigenvalues. For this reason, if we use $k < p$ principal components, the reduced (compressed) matrix is of order $n \times k$ and is calculated as

$$\boldsymbol{X}^{*} = \boldsymbol{X}\boldsymbol{W}^{*},$$

where $\boldsymbol{W}^{*}$ contains the same rows of $\boldsymbol{W}$, but only the first k columns instead of the original p columns. The selection of the number of principal components to maintain is critical, and we therefore provide some classical rules for this process:

(a) Select the required principal components to cover a certain amount of variances, such as 80% or 90%.
(b) Order the eigenvalues from highest to lowest, then make a plot of each of the ordered eigenvalues against its position and select as the number of principal components that number from which little variance is gained by retaining additional eigenvalues. This plot is called a scree plot.
(c) Discard those components associated with eigenvalues below a certain level, which is usually set as the average variance. In particular, when working with a correlation matrix built with the input matrix, $\boldsymbol{X}$, the average value of the components is 1, and this rule leads to selecting eigenvalues greater than the unit.

It is important to point out that the principal components can be obtained from a covariance matrix, $\boldsymbol{Q} = \frac{1}{n-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, where each column of $\boldsymbol{X}$ is centered, or from the correlation matrix, $\boldsymbol{Q} = \frac{1}{n-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, where each column of the original matrix of information was standardized. The covariance matrix is used to calculate the principal components when all the independent variables were measured using the same scale, but if they were measured in different scales, we recommend calculating the principal components with the correlation matrix, which agrees with the normalization methods that are suggested when the independent variables were measured in different scales.

Assume that we measured 15 observations (lines) and five independent variables, and the collected data is given in Table 2.13.

Then we place these data (Table 2.13) in a data frame we called Data. Since the data are in different scales, each column is standardized using the function scale in R, and the first six observations of the scaled variables are given below. The complete code that provides the output given below is available in Appendix 2.

```
> Data=read.csv("Simulated_PCA.csv", header = T)
>
> ####We scale each column of the predictors
> Rscaled=scale(Data[,-1])
> head(Rscaled)
```

Table 2.13 Five independent variables in different scales

Line	Yield	PlantHeight	DaysFlowering	DaysMaturity	WeightFreshPlant
L1	5.49	179.98	64.6	119.39	43.86
L2	6.84	181.1	64.68	121.67	44.72
L3	6.75	181	64.38	120.1	45.36
L4	4.98	180.41	64.03	120.47	44.21
L5	8.36	180.89	66.13	122.3	45.42
L6	4.43	179.74	63.26	119.88	43.65
L7	6.67	180.49	64.83	120.22	44
L8	4.44	177.94	62.31	118.46	42.6
L9	5.62	178.91	63.41	120.64	44.07
L10	6.96	179.93	64.03	119.99	43.53
L11	5.91	181.21	64.03	120.86	43.82
L12	5.59	180.32	64.89	120.98	45.21
L13	5.27	180.97	63.75	120.14	44.29
L14	4.32	177.79	62.47	118.72	42.28
L15	5.48	180.04	63.82	120.08	43.86

```
      Yield PlantHeight DaysFlowering DaysMaturity WeightFreshPlant
[1,] -0.2814557   -0.06307151    0.57645346  -0.8738660    -0.2214903
[2,]  0.9159136    0.97575340    0.65900049   1.4162657     0.7373101
[3,]  0.8360890    0.88300118    0.34944911  -0.1607110     1.4508360
[4,] -0.7337952    0.33576305   -0.01169416   0.2109332     0.1687191
[5,]  2.2640628    0.78097373    2.15516550   2.0490652     1.5177291
[6,] -1.2216124   -0.28567685   -0.80620937  -0.3816886    -0.4556160
```

We then calculate the scaled variance (correlation) $\boldsymbol{Q} = \frac{1}{n-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ and from this matrix we calculate the eigenvalue decomposition using the function eigen() of R. Next, we extract the eigenvectors and eigenvalues, and using the eigenvalue information, we calculate the variance of each principal component as shown below.

```
> n=nrow(Rscaled)
> Q_scaled=(t(Rscaled)%*%Rscaled)/(n-1) #Q_scaled=var(Rscaled)
> ####Eigenvalue decomposition
> SVD_Rscaled=eigen(Q_scaled)
>
> ####Extracting eigenvectors and eigenvalues
> EVectors=SVD_Rscaled$vectors
> Eigenvalues=SVD_Rscaled$values
> Standar.deviations.PC=sqrt(Eigenvalues)
> Standar.deviations.PC
[1] 2.0090648 0.6469991 0.4964878 0.4356803 0.3297472
```

Afterward, we manually calculate the principal components using the expression PC $= \boldsymbol{XW}$, where $\boldsymbol{W}$ is the matrix of eigenvectors extracted in the previous code and

denoted as EVectors. The calculation used here was the scaled matrix (Rscaled) instead of the original matrix of independent variables.

```
> ####Principal components for all the p=5 variables
> PCM=Rscaled%*%EVectors
> head(PCM)
      [,1]           [,2]          [,3]        [,4]         [,5]
[1,] -0.371645755   0.02096745  0.6294630  0.59140700  -0.58621466
[2,]  2.096405335 -0.02185667 -0.2605723 -0.54289807   0.12914892
[3,]  1.489539155 -0.30245825  0.5022823  0.73805410   0.79018009
[4,] -0.001842872 -0.78507631 -0.2113135 -0.06274286  -0.24361792
[5,]  3.931856138  1.09587334 -0.4321212  0.05736360  -0.17988334
[6,] -1.403180027 -0.72567496 -0.2400658 -0.12744453  -0.08832723
```

We then built the scree plot, which is one of the three tools used to select the number of principal components to maintain.

```
> ####Scree plot
> Ordered_Eigenvalues=sort(Eigenvalues,decreasing =T )
> plot(Ordered_Eigenvalues, type = "l",ylab="Variances",
xlab="Principal components")
```

Figure 2.2 shows that after two principal components, there are no significant gains in variance explained by adding more principal components. Due to this, we can select the first two principal components that explain 89.09% of the total variance of the complete data set.

Finally, the next part of the code selects only the first two principal components that will replace the whole matrix of scaled independent variables denoted here as Rscaled. Therefore, the selection process only consists of extracting the first two columns of the matrix that contain all the principal components, as shown in the following code. This reduced matrix is then replaced by the original matrix of independent variables as input in any statistical machine learning algorithm.

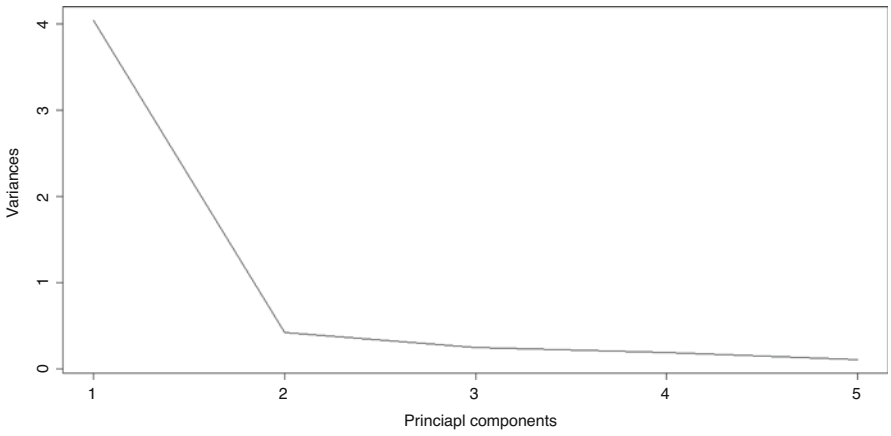

Fig. 2.2 Scree plot for the five independent variables in different scales

```
> ####Principal components for the first k=2 principal components
> X_star=PCM[,1:2]
> head(X_star)
       [,1]          [,2]
[1,]  -0.371645755  0.02096745
[2,]   2.096405335 -0.02185667
[3,]   1.489539155 -0.30245825
[4,]  -0.001842872 -0.78507631
[5,]   3.931856138  1.09587334
[6,]  -1.403180027 -0.72567496
```

It is important to point out that the code given above was used to manually calculate the new variables, called principal components. However, there are also many libraries of R that can be used to carry out this process automatically. One of these functions is the prcomp(), which performs a principal component analysis when the only input provided is the original scaled (or not scaled) matrix of original inputs. Next, we show how to use this function to perform a principal component analysis. Then, with the function summary(), the standard deviation of each of the five principal components is extracted, as well as the proportion of variance that explains each principal component and the cumulative proportion of variance explained for the principal component. Here, we can see that the standard deviations obtained using this function are the same as those obtained above when the principal components were extracted manually using the eigen() function.

```
> ####PCA analysis using the function prcomp
> PCA=prcomp(Rscaled)
> summary(PCA)
Importance of components:
                          PC1     PC2     PC3     PC4     PC5
Standard deviation     2.0091  0.64700  0.4965  0.43568  0.32975
Proportion of Variance 0.8073  0.08372  0.0493  0.03796  0.02175
Cumulative Proportion  0.8073  0.89099  0.9403  0.97825  1.00000
```

We then show how to extract all the principal components resulting from using the prcomp() function. The principal components extracted are clearly the same, except with negative values, which is not a problem since this does not alter the solution.

```
> ###Extracting the principal components####
> PC_All=PCA$x
> head(PCA$x)
       PC1          PC2          PC3         PC4          PC5
[1,] -0.371645755 -0.02096745  0.6294630 -0.59140700 -0.58621466
[2,]  2.096405335  0.02185667 -0.2605723  0.54289807  0.12914892
[3,]  1.489539155  0.30245825  0.5022823 -0.73805410  0.79018009
[4,] -0.001842872  0.78507631 -0.2113135  0.06274286 -0.24361792
[5,]  3.931856138 -1.09587334 -0.4321212 -0.05736360 -0.17988334
[6,] -1.403180027  0.72567496 -0.2400658  0.12744453 -0.08832723
```

Finally, we make the scree plot using the output of the prcomp() function, and the output of this figure is exactly the same as that given in Fig. 2.2.

```
> #####Variances of each principal component and scree plot####
> Var_PC=PCA$sdev*PCA$sdev
> plot(Var_PC, type = "l",ylab="Variances", xlab="Principal
components")
```

Appendix 1

```
rm()
library(synbreed)
####Loading the marker information
snp7 <- read.csv("MarkersToy.csv",header=T)
snp7=snp7[,-1]
snp7

####Set names for individuals
rownames(snp7) <- paste("ID",1:8,sep="")
pos.NA=which(snp7=="?_?", arr.ind=TRUE)
snp7[pos.NA]=NA
snp7
####Creating an object of class 'gpData'
gp <- create.gpData(geno=snp7)

####Recoding to 0, 1, and 2 values the genotypic data
gp.coded <-codeGeno(gp)
Geno_Recoded=gp.coded$geno
Geno_Recoded

####Recoding to values of 0, 1, and 2 the genotypic data and inputting
Imputed_Geno<-codeGeno(gp,impute=T,impute.type="random")
Imputed_Geno$geno
```

Appendix 2

```
rm(list=ls())
Data=read.csv("Simulated_PCA.csv", header = T)

####We scale each column of the predictors
Rscaled=scale(Data[,-1])
head(Rscaled)
n=nrow(Rscaled)
Q_scaled=(t(Rscaled)%*%Rscaled)/(n-1) #Q_scaled=var(Rscaled)
####Eigenvalue decomposition
SVD_Rscaled=eigen(Q_scaled)
```

```
####Extracting eigenvectors and eigenvalues
EVectors=SVD_Rscaled$vectors
Eigenvalues=SVD_Rscaled$values
Standar.deviations.PC=sqrt(Eigenvalues)
Standar.deviations.PC

####Principal components for all the p=5 variables
PCM=Rscaled%*%EVectors
head(PCM)

####Scree plot
Ordered_Eigenvalues=sort(Eigenvalues,decreasing =T )
plot(Ordered_Eigenvalues, type = "l",ylab="Variances",
xlab="Principal components")

####Principal components for the first k=2 principal components
X_star=PCM[,1:2]
head(X_star)

####PCA analysis using the function prcomp
PCA=prcomp(Rscaled)
summary(PCA)

###Extracting the principal components####
PC_All=PCA$x
head(PCA$x)

#####Variances of each principal component and scree plot####
Var_PC=PCA$sdev*PCA$sdev
plot(Var_PC, type = "l",ylab="Variances", xlab="Principal
components")
```

References

Austin PC, Brunner LJ (2004) Inflation of the type I error rate when a continuous confounding variable is categorized in logistic regression analyses. Stat Med 23(7):1159–1178

Griffiths JF, Griffiths AJ, Wessler SR, Lewontin RC, Gelbart WM, Suzuki DT, Miller JH (2005) An introduction to genetic analysis. Macmillan, New York

Henderson CR (1950) Estimation of genetic parameters. Ann Math Stat 21:309–310

Henderson CR (1963) Selection index and expected genetic advance. In: Hanson WD, Robinson HF (eds) Statistical genetics and plant breeding, Publication 982. Washington, DC, National Academy of Sciences, National Research Council, pp 141–163

Henderson CR (1973) Sire evaluation and genetic trends. In: Proceedings of the animal breeding and genetics symposium in honor of J. L. Lush. Blackburgh, Champaign, IL, American Society for Animal Science, pp 10–41

Henderson CR (1975) Best linear unbiased estimation and prediction under a selection model. Biometrics 31:423–447

Henderson CR (1984) Applications of linear models in animal breeding. University of Guelph, Guelph

Kuhn M, Johnson K (2013) Applied predictive modeling. Springer, New York
Littell RC, Milliken GA, Stroup WW, Wolfinger RD (1996) SAS for mixed models. SAS Institute, Inc., Cary, NC
Littell RC, Milliken GA, Stroup WW, Schabenberger O, Wolfinger RD (2006) SAS for mixed models, 2nd edn. SAS Institute, Cary, NC
Marsjan PA, Oldenbroek JK (2007) Molecular markers, a tool for exploring genetic diversity. In: The state of the world's animal genetic resources for food and agriculture. FAO, Rome. ISBN 9789251057629
Piepho HP, Möhring J, Melchinger AE, Büchse A (2008) BLUP for phenotypic selection in plant breeding and variety testing. Euphytica 161:209–228. https://doi.org/10.1007/s10681-007-9449-8
Schork NJ, Fallin D, Lanchbury S (2000) Single nucleotide polymorphisms and the future of genetic epidemiology. Clin Genet 58:250–264
Searle SR, Casella G, McCulloch CE (2006) Variance components. Wiley, Hoboken, NJ
Stonecking M (2001) From the evolutionary past. Nature 409:821–822
Stroup WW (2012) Generalized linear mixed models: modern concepts, methods and applications. CRC Press, Boca Raton, FL
The International SNP Map Working Group (2001) A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. Nature 409:928–933
VanRaden PM (2008) Efficient methods to compute genomic predictions. J Dairy Sci 91:4414–4423
Zeger SL, Liang KY, Albert PS (1998) Models for longitudinal data: a generalized estimating equation approach. Biometrics 44(4):1049–1060

Chapter 3
Elements for Building Supervised Statistical Machine Learning Models

3.1 Definition of a Linear Multiple Regression Model

A linear multiple regression model (LMRM) is a useful tool for investigating linear relationships between two or more explanatory variables (inputs, features in machine learning literature) (X) and the conditional expected value of a response $E(Y/X)$. Due to its simplicity, adequate fitting, and easily interpretable results, this has been one of the most popular techniques for studying the association between variables. Specifically, regarding the latter task, this is a useful approach and an ideal (natural) starting point for studying more advanced methods (James et al. 2013) of association and prediction.

In this chapter, we review the main concepts and approaches for fitting a linear regression model.

3.2 Fitting a Linear Multiple Regression Model via the Ordinary Least Square (OLS) Method

In a general context, we have a covariate vector $X = (X_1, \ldots, X_p)^{\mathrm{T}}$ and we want to use this information to predict or explain how this variable affects a real-value response Y. The linear multiple regression model assumes a relationship given by

$$Y = \beta_0 + \sum_{j=1}^{p} X_j \beta_j + \epsilon, \tag{3.1}$$

where ϵ is a random error with mean 0, $E(\epsilon) = 0$ and is independent of X. This error is included in the model to capture measurement errors and the effects of other unregistered explanatory variables that can help to explain the mean response.

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_3

Then, the conditional mean of this model is $E(Y|X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$ and the conditional distribution of Y given X is only affected by the information of X.

For estimating the parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^{\mathrm{T}}$, usually we have a set of data $\left(\boldsymbol{x_i}^{\mathrm{T}}, y_i\right)$, $i = 1, \ldots, n$, often known as training data, where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}}$ is a vector of features measurement and y_i is the response measurement corresponding to the ith individual drawn. The most common method for estimating $\boldsymbol{\beta}$ is the least squares method (OLS) that consists of taking the $\boldsymbol{\beta}$ value that minimizes the residual sum of squares defined as

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{\beta}_0\right)^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}),$$

where $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_p)^{\mathrm{T}}$, $\boldsymbol{y} = (y_1, \ldots, y_n)^{\mathrm{T}}$ is the vector with the response values of all individuals, and $\boldsymbol{X}$ is an $n \times (p + 1)$ matrix that contains the information of the measured features of all individuals, including the intercept in the first entry:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix}.$$

If the $\boldsymbol{X}$ matrix has full column rank, then by differentiating the residual sum of squares with respect to the $\boldsymbol{\beta}$ coefficients, we can find the set of $\boldsymbol{\beta}$ parameters that minimize the $\mathrm{RSS}(\boldsymbol{\beta})$,

$$\frac{\mathrm{RSS}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \frac{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \frac{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{2y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\beta}^{\mathrm{T}}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{\beta}}{\partial \boldsymbol{\beta}} = \boldsymbol{2}\left[\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{\beta} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}\right]$$

This derivative is also known as the gradient of the residual sum of squares. Then by setting the gradient of the residual sum of squares to zero, we obtain the normal equations

$$\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{\beta} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}$$

The solution to the normal equations is unique and gives the OLS estimator of $\boldsymbol{\beta}$

$$\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y},$$

where super index -1 indicates the inversion matrix.

From the above assumptions, we can show that this estimator is unbiased

$$\begin{aligned} E\left(\widehat{\boldsymbol{\beta}}\right) &= E\left[\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\right] = E\left[\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{\beta} + \epsilon)\right] \\ &= E\left[\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta}\right] + \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}E(\epsilon) = \boldsymbol{\beta}. \end{aligned}$$

and with the additional assumption that the observation responses y_i's are uncorrelated and have the same variance, $\text{Var}(y_i) = \sigma^2$, we can also show that the variance–covariance matrix of this is

$$\text{Var}\left(\widehat{\boldsymbol{\beta}}\right) = \sigma^2 \left(\boldsymbol{X}^{\text{T}}\boldsymbol{X}\right)^{-1}.$$

When the input features only contain the information of a variable ($p = 1$), the resulting model is known as simple linear regression and can be easily visualized in the Cartesian plane. When $p = 2$, the above multiple linear regression describes a plane in the three-dimensional space (x_1, x_2, y). In general, the conditional expected value of this model defines a hyperplane in the p-dimensional space of the input variables (Montgomery et al. 2012).

The fitted values corresponding to all the training individuals are

$$\widehat{\boldsymbol{y}} = \boldsymbol{X}\widehat{\boldsymbol{\beta}} = \boldsymbol{X}\left(\boldsymbol{X}^{\text{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\text{T}}\boldsymbol{y} = \boldsymbol{H}\boldsymbol{y},$$

where the matrix $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^{\text{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\text{T}}$ is commonly called the hat matrix. This is because the vector of the observed response values is mapped by this expression to a vector of fitted values (Montgomery et al. 2012), in this way, puts the hat on $\boldsymbol{y}$ (Hastie et al. 2009). In a similar way, a predicted value of an arbitrary individual with feature $\boldsymbol{x}$ can be obtained by

$$\widehat{\boldsymbol{y}}^{*} = \boldsymbol{x}^{*\text{T}}\widehat{\boldsymbol{\beta}},$$

where $\boldsymbol{x}^{*} = (1, \boldsymbol{x}^{\text{T}})^{\text{T}}$.

An unbiased estimator for the common residual variance σ^2 is obtained by

$$\begin{aligned}
\widehat{\sigma}^2 &= \frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2 \\
&= \frac{1}{n-p-1}\sum_{i=1}^{n} e_i^2 \\
&= \frac{1}{n-p-1}\boldsymbol{e}^{\text{T}}\boldsymbol{e} \\
&= \frac{1}{n-p-1}(\boldsymbol{y} - \widehat{\boldsymbol{y}})^{\text{T}}(\boldsymbol{y} - \widehat{\boldsymbol{y}}) \\
&= \frac{1}{n-p-1}\boldsymbol{y}^{\text{T}}(\boldsymbol{I}_n - \boldsymbol{H})\boldsymbol{y},
\end{aligned}$$

where $e_i = y_i - \widehat{y}_i$ is known as the residual of the model corresponding to the individual i, $\boldsymbol{e} = \boldsymbol{y} - \widehat{\boldsymbol{y}}$ is the vector of all residual values, and $\boldsymbol{I}_n$ is the identity matrix of order $n \times n$.

The traditional inferential and prediction analysis for this model assumes that the random error ϵ is normally distributed with mean zero and variance σ^2. With this we

can show that the OLS of beta coefficients, $\widehat{\boldsymbol{\beta}}$, is a random vector distributed according to a multivariate normal distribution with vector mean $\boldsymbol{\beta}$ and a variance–covariance matrix, as previously defined (Montgomery et al. 2012; Hastie et al. 2009; Rencher and Schaalje 2008). Another important fact that will be described in more detail in the next section, is that under the Gaussian assumption over errors, the OLS of $\boldsymbol{\beta}$ coincides with the maximum likelihood estimator.

We can also show that $(n-p-1)\widehat{\sigma}^2/\sigma^2$ is independent of $\widehat{\boldsymbol{\beta}}$ and distributed according to a Chi-squared distribution with $n-p-1$ degrees of freedom. Based on this and on the properties of the normal and t-student distributions, we show that for each $j=0,\ldots,p$, $T_j = \left(\widehat{\beta}_j - \beta_j\right)/\sqrt{c_{jj}\widehat{\sigma}^2}$, where c_{jj} is the $(j+1, j+1)$ elements of the matrix $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}$, are random variables with a t-student distribution with $n-p-1$ degrees of freedom (t_{n-p-1}). That is, $T_j \sim t_{n-p-1}$ and $\sim$ stands for distributed as. From here, a $100(1-\alpha)\%$ confidence interval for a particular beta coefficient, β_j, is given by

$$\widehat{\beta}_j \pm t_{1-\alpha/2,\ n-p-1}\sqrt{c_{jj}\widehat{\sigma}^2},$$

where $t_{\alpha,\, n-p-1}$ is the α quantile of the t-student distribution with $n-p-1$ degrees of freedom. Similarly, a $100(1-\alpha)\%$ joint confidence region for all the beta coefficients, $\boldsymbol{\beta}$, is given if these values satisfy

$$\frac{\left(\widehat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\left(\widehat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right)}{(p+1)\widehat{\sigma}^2} \leq F^{p+1}_{1-\alpha,\ n-p-1},$$

where $F^{p+1}_{\alpha,n-p-1}$ denotes the α quantile of the F distribution with $p+1$ and $n-p-1$ degrees of freedom in the numerator and denominator, respectively (Rencher and Schaalje 2008).

In a similar way, to test a hypothesis over a specific beta coefficient, $H_{0j}=\beta_j=\beta_{j0}$, the following rule can be used: reject H_{0j} if $T_{j0} = \left(\widehat{\beta}_j - \beta_{j0}\right)/\sqrt{c_{jj}\widehat{\sigma}^2}$ is "large" in magnitude, that is, if $|T_{j0}| > t_{1-\alpha/2,\, n-p-1}$, where α is the desired level test. More generally, the test $H_0 = \boldsymbol{W\beta} = \boldsymbol{w}$, where $\boldsymbol{W}$ is a $q \times (p+1)$ matrix of rank $q \leq p+1$, can be performed using the following rule:

$$\text{reject } H_0 \text{ if } F = \frac{n-p-1}{q}\,\frac{(\boldsymbol{W\beta}-\boldsymbol{w})^{\mathrm{T}}\left[\boldsymbol{W}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{W}^{\mathrm{T}}\right]^{-1}(\boldsymbol{W\beta}-\boldsymbol{w})}{\widehat{\sigma}^2} \geq F^{p+1}_{1-\alpha,n-p-1}.$$

3.3 Fitting the Linear Multiple Regression Model via the Maximum Likelihood (ML) Method

The maximum likelihood (ML) estimation is a more general and popular method for estimating the parameters of a model (Casella and Berger 2002). It consists of finding the parameter value that maximizes the "probability" of observed values in the sample under the adopted model. Specifically, if $\left(\boldsymbol{x}_i^{\mathrm{T}}, y_i\right)$, $i = 1, \ldots, n$, is a set of observations from a multiple linear regression model (3.1) with homoscedastic and uncorrelated errors, the MLE of $\boldsymbol{\beta}$ and σ^2, $\widehat{\boldsymbol{\beta}}$ and $\widehat{\sigma}^2$, of this model is defined as

$$\left(\widehat{\boldsymbol{\beta}}^{\mathrm{T}}, \widehat{\sigma}^2\right) = \arg\max_{\boldsymbol{\beta}, \sigma^2} L\left(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right),$$

where $L(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X})$ is the likelihood function of the parameters, which is the probability of the observed response values but viewed as a function of the parameters

$$L\left(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right].$$

Then, the $\log(L(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}))$ is equal to

$$\log\left(L\left(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right)\right) = -\frac{n}{2}\log(2\pi) - n\log(\sigma) - \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

To find the maximum of σ^2 and $\boldsymbol{\beta}$, we get the derivative of $\log\left(L\left(\widehat{\boldsymbol{\beta}}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right)\right)$ with regard to these parameters

$$\frac{\log\left(L(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X})\right)}{\partial\boldsymbol{\beta}} = \frac{\left[\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{\beta} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}\right]}{\sigma^2}$$

$$\frac{\log\left(L(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X})\right)}{\partial\sigma^2} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

Now, by setting these derivatives equal to zero and solving the resulting equations for $\boldsymbol{\beta}$ and σ^2, we found that the estimates of these parameters are

$$\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

$$\widehat{\sigma}^2 = \frac{1}{n}\left(\boldsymbol{y} - \boldsymbol{X}\widehat{\boldsymbol{\beta}}\right)^{\mathrm{T}}\left(\boldsymbol{y} - \boldsymbol{X}\widehat{\boldsymbol{\beta}}\right).$$

From this we can see that for each value of σ^2, the value of $\boldsymbol{\beta}$ that maximizes the likelihood is the same value that maximizes $-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$, which in turn

minimizes $(\boldsymbol{y}-\boldsymbol{X\beta})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X\beta})$, which is precisely the OLS of $\boldsymbol{\beta}, \widehat{\boldsymbol{\beta}}$. But when equating the derivative of $\log\left(L\left(\widehat{\boldsymbol{\beta}}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right)\right)$ to zero and solving for σ^2, the value of σ^2 that maximizes $L\left(\widehat{\boldsymbol{\beta}}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right)$ is $\widehat{\sigma}^2 = \frac{1}{n}\left(\boldsymbol{y}-\boldsymbol{X}\widehat{\boldsymbol{\beta}}\right)^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{X}\widehat{\boldsymbol{\beta}}\right)$.

Finally,

$$L(\boldsymbol{\beta}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}) \leq L\left(\widehat{\boldsymbol{\beta}}, \sigma^2; \boldsymbol{y}, \boldsymbol{X}\right) \leq L\left(\widehat{\boldsymbol{\beta}}, \widehat{\sigma}^2; \boldsymbol{y}, \boldsymbol{X}\right)$$

and from here, the MLE of $\boldsymbol{\beta}$ and σ^2 are $\widehat{\boldsymbol{\beta}}$ and $\widehat{\sigma}^2$, because it can be shown that the values of parameters that maximize the likelihood are unique when the design matrix $\boldsymbol{X}$ is of full column rank.

3.4 Fitting the Linear Multiple Regression Model via the Gradient Descent (GD) Method

The steepest descent method, also known as the gradient descent (GD) method, is a first-order iterative algorithm for minimizing a function (f). It is a central mechanism in statistical learning to training models (to estimate the parameters), for example, in neuronal networks and penalized regression models (Ridge and Lasso). It consists of successively updating the argument of the objective function in the direction of the steepest descent (along the negative of the gradient of the function), that is, in the direction in which f decreases most rapidly (Haykin 2009; Nocedal and Wright 2006). Specifically, each step of this algorithm is described by

$$\eta_{t+1} = \eta_t - \alpha \nabla f(\eta_t),$$

where $\nabla f(\eta_t)$ is the gradient vector of f evaluated in the current value η_t and α is a step size or learning rate parameter, which greatly determines the convergence behavior toward an optimal solution (Haykin 2009; Beysolow II 2017) and in neural networks it is popular for setting this at a small, fixed value (Warner and Misra 1996; Goodfellow et al. 2016). The learning rate parameter can be adaptative as well, that is, can be allowed to change at each step. For example, in the library Keras (see Chap. 11) that can be used for implementing and training neuronal networks models, there are several optimizers based on an adaptive gradient descendent algorithm such as Adam Adgrad, Adadelta, RMSprop, among others (Allaire and Chollet 2019). The ideal value of the step size would be the value that gives the larger reduction in each step, that is, the value of α that minimizes $f(\eta_t - \alpha \nabla f(\eta_t))$, which in general is difficult and expensive to obtain (Nocedal and Wright 2006).

Although the use of this algorithm could be avoided in an MLR, especially in small data sets, and also because of its slow convergence in linear systems (Burden and Faires 2011), here we will describe how this works when finding the optimal

beta coefficients in this model. First, the gradient of the residual sum of squares is given by

$$\nabla \mathrm{RSS}(\boldsymbol{\beta}) = 2\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\right).$$

Then, the next update of beta coefficients in the gradient descent algorithm in this model is given by

$$\begin{aligned}\boldsymbol{\beta}_{t+1} &= \boldsymbol{\beta}_t - 2\alpha\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta}_t - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\right)\\ &= \boldsymbol{\beta}_t - 2\alpha\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{\beta}_t - \boldsymbol{y})\\ &= \boldsymbol{\beta}_t + 2\alpha\boldsymbol{X}^{\mathrm{T}}\boldsymbol{e}_t,\end{aligned}$$

where $\boldsymbol{e}_t = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_t$ is the vector of residuals that is obtained in the current iteration. One way to speed up the convergence of the algorithm is by choosing the ideal learning rate in each step, which, as was described before, is given by the value of α that minimizes $f(\eta_t - \alpha \nabla f(\eta_t))$, and in this case for the MLR model is given by (Nocedal and Wright 2006):

$$\alpha_t = \frac{\boldsymbol{e}_t^{\mathrm{T}}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{e}_t}{\boldsymbol{e}_t^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)^2\boldsymbol{e}_t}.$$

Example 1 For numerical illustration, we considered a synthetic data set that consists of 100 observations and two covariates. The scatter plots in Fig. 3.1 show how the response variable (y) is related to the two covariates (x_1, x_2). By setting a value of 10^{-2} for the learning rate parameter, and as the stopping criterion a tolerance of 10^{-8} for the maximum norm of the difference between the current and next vector value, the beta coefficient obtained with the GD method is $\widehat{\boldsymbol{\beta}} = (5.0460764, 0.8551383, 2.1903356)$. For these synthetic examples, 12 iterations were necessary, while by changing the learning rate parameter to 10^{-3}, the number of iterations increased to 185, but we practically got the same results. Now, by using the "optimal" learning rate parameter described before for MLR with the same tolerance error (10^{-8}), the number of required iterations up to convergence is reduced to only 10 iterations. In general, the performance of the gradient descent depends greatly on the objective function and can be affected by the characteristics of the model, the dispersion of the data (explained variance of the predictors), and the dependence between the predictors, among others.

In the data set used in this example, the covariates are independent and the proportion of explained variances by the predictor is about 79% of the total variance of the response. By changing to a pair of moderately correlated covariates with correlation 0.75, while holding the same beta coefficient values, the variance of the residual (1.44), and the same sample size, we generate data where a greater proportion of variance is explained by the covariates (85.6%), but when applying the

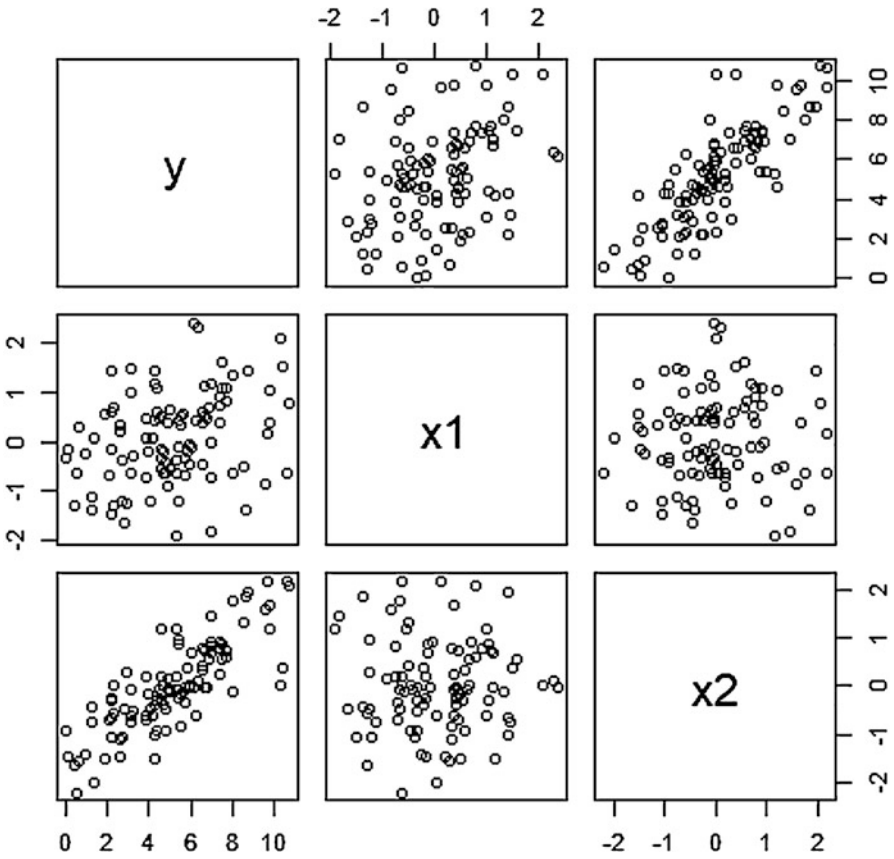

Fig. 3.1 Scatter plot of synthetic data generated from an MLR with two covariates

gradient descent with the same tolerance error (10^{-8}) as before, and learning rate values of 10^{-2} and 10^{-3}, the required number of iterations are about 5.75 times (69) and 3.5 times (649) the number required for the independent covariates case and the example described before, respectively.

Continuing with the last case of dependent variables, when using the optimal learning rate described before for the MLR, the number of iterations is reduced to 60, 9 less than when using the constant learning rate 10^{-2}.

By multiplying the beta coefficients used before by sqrt(0.1), the proportion of explained variance by the covariates is reduced to 27.30% and 37.2% in the same independent covariate (E3) and the same correlated covariate (E4) scenarios described before, respectively. With a tolerance error of 10^{-8} and with a learning rate equal to 10^{-2}, the required number of iterations are 183 and 66, for scenarios E3 and E4, respectively, while for a learning rate of 10^{-3} the required number of iterations are 617 and 1638. When using the "optimal" learning rate parameter, the required number of iterations is reduced to 17 and 56 for scenarios E3 and E4, respectively.

The R code used for implementing the GD method is given next.

```
##################R code for Example 1 ################################
rm(list=ls())
library(mvtnorm)
set.seed(1)
X = cbind(1,rmvnorm(100,c(0,0),diag(2)))
#Uncomment the next three lines code to simulate dependent covariables
#Sigma_X =0.75+0.25*diag(2)
#L = t(chol(Sigma_X))
#X = X%*%t(L)

betav = c(5,1,2.1)
#Uncomment the next line code to reduce the value of the beta cofficients
and reduce the proportion of variance of the response explained by the
features
betav = sqrt(0.1)*beta
y = X%*%betav + rnorm(100,0,1.2)
dat = data.frame(y=y,x1 = X[,2],x2=X[,3])
plot(dat)

alpha = 1e-2
#alpha = 1e-3
tol   = 1e-8
p = 2
betav_0 = c(mean(y),rep(0,p))
tol.e = 1
Iter = 0
tX = t(X)
XtX = X%*%t(X)
while(tol<tol.e)
{
 Iter = Iter + 1
 e = y-X%*%betav_0
 #Uncomment the next line code to use the optimal learning rate
 #alpha = (t(e)%*%XtX%*%e/(t(e)%*%(XtX)%*%XtX%*%e))[1,1]
 betav_t = betav_0 + alpha*tX%*%e
 tol.e = max(abs(betav_t-betav_0))
 betav_0 = betav_t
}
betav_t
tol.e
Iter
```

This code is only for illustrative purposes, that is, to illustrate in a very transparent way how the GD method can be implemented. Of course the existing statistical machine learning software programs implement this method and so there is no need to use this program for real applications, since the existing software programs that implement this method do a lot of work more efficiently and in a more user-friendly way.

3.5 Advantages and Disadvantages of Standard Linear Regression Models (OLS and MLR)

The MLR is a simple and computationally appealing class of models, but with many predictors (relative to the sample size) or nearly dependent features, it may result in large prediction error and/or large predictive intervals (Wakefield 2013). To appreciate the latter case (nearly dependent features), consider the spectral decomposition $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^{\mathrm{T}}$, where $\boldsymbol{\Lambda} = \mathrm{Diag}(\lambda_0, \ldots, \lambda_p)$ is a diagonal matrix with the eigenvalues of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ in decreasing order and $\boldsymbol{\Gamma}$ is an orthogonal matrix with columns corresponding to eigenvectors of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. Then the obtained variance–covariance matrix of the OLS estimator of $\widehat{\boldsymbol{\beta}}$ can be expressed as

$$\mathrm{Var}\left(\widehat{\boldsymbol{\beta}}\right) = \sigma^2\left(\boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^{\mathrm{T}}\right)^{-1} = \sigma^2\boldsymbol{\Gamma}\boldsymbol{\Lambda}^{-1}\boldsymbol{\Gamma}^{\mathrm{T}}.$$

When the features are nearly dependent, some λ_j's will be "close" to zero and consequently the variance of some $\widehat{\beta}_j'$s will be high; this is even greater when the linear dependence of the features is strong (Wakefield 2013; Christensen 2011). This strong dependence between features is a problem of the OLS in MLR that is also reflected in the quality of the prediction performance, for example, when this is measured by the conditional expected prediction error (EPE) or mean squared error prediction that for an individual with feature $\boldsymbol{x}_o$ is given by

$$\begin{aligned}
\mathrm{EPE}(\boldsymbol{x}_o) &= E_{\boldsymbol{Y},Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(Y_o - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}\right)^2\right] \\
&= E_{\boldsymbol{Y},Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left\{\left[\left(Y_o - E(Y_0|\boldsymbol{x}_0) + E(Y_0|\boldsymbol{x}_0) - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}\right)\right]^2\right\} \\
&= E_{\boldsymbol{Y},Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(Y_o - E\left(Y_0|\boldsymbol{x}_0^{*\mathrm{T}}\right)\right)^2\right] + 2E_{Y_o|\boldsymbol{x}_o}\left[\left(Y_o - E\left(Y_0|\boldsymbol{x}_0^{*\mathrm{T}}\right)\right)\right]\left[E\left(Y_0|\boldsymbol{x}_0^{*\mathrm{T}}\right) - E_{Y|X}\left(\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}\right)\right] \\
&\quad + E_{\boldsymbol{Y},Y_o|\boldsymbol{X}}\left[\left(E\left(Y_0|\boldsymbol{x}_0^{*\mathrm{T}}\right) - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}\right)^2\right] \\
&= \sigma^2 + E_{\boldsymbol{Y},Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\beta} - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}\right)^2\right] = \sigma^2 + Var\left(\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}|\boldsymbol{x}_o\right) \\
&= \sigma^2 + \sigma^2\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\Gamma}\boldsymbol{\Lambda}^{-1}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{x}_o^* \\
&= \sigma^2\left(1 + \sum_{j=0}^{p}\frac{\left(x_{oj}^{**}\right)^2}{\lambda_j}\right),
\end{aligned}$$

where $\boldsymbol{x}_o^{**} = \boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{x}_o^* = \left(x_{o0}^{**}, \ldots, x_{op}^{**}\right)^{\mathrm{T}}$. This means that the average loss incurred (squared difference between the value to be predicted and the predicted value) by predicting Y_0 with its estimated mean under the MLR, $\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}$, is composed of intrinsic or irreducible data noise (first term) and the variance of $\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}$ (second term). The former cannot be avoided no matter how well the mean value of $Y_0 \mid x_0$, $E(Y_0|\, x_0)$, is

estimated, and the latter increases as the dependence of features is stronger. From this, it is apparent that the EPE is also affected by the strong dependence between features, which is a problem of the OLS in an MLR in a prediction context.

3.6 Regularized Linear Multiple Regression Model

3.6.1 Ridge Regression

Ridge regression, originally proposed as a method to combat multicollinearity, is also a common approach for controlling overfitting in an MLR model (Christensen 2011). It translates the OLS problem into the minimization of the penalized residual sum of squares defined as

$$\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta}) = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2,$$

where $\lambda \geq 0$ is known as the regularization or tuning parameter, which determines the level or degree to which the beta coefficients are shrunk toward zero. When $\lambda = 0$, the OLS is the solution to the beta coefficients, but when λ is large, the $\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta})$ is dominated by the penalization term, and the OLS solution has to shrink toward 0 (Christensen 2011). In general, when the number of parameters to be estimated is larger than the number of observations, the estimator can be highly variable. In this situation, the intuition of Ridge regression tries to alleviate this by constraining the sum of squares for the beta coefficients.

Note that $\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta})$ can be expressed as

$$\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta}) = \mathrm{RSS}(\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{\beta},$$

where $\boldsymbol{D} =$ diag $(0, 1, \ldots, 1)$ is an identity matrix of dimension $(p + 1) \times (p + 1)$ but with one zero in its first entry. Then, the gradient of $\mathrm{RSS}_{\lambda}(\boldsymbol{\beta})$, that is, the first derivative with regard to $\boldsymbol{\beta}$ of $\mathrm{RSS}_{\lambda}(\boldsymbol{\beta})$, is

$$\nabla\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta}) = 2\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\right) + 2\lambda\boldsymbol{D}\boldsymbol{\beta}.$$

Solving $\nabla\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta}) = \mathbf{0}$, the Ridge solution is given by

$$\widehat{\boldsymbol{\beta}}^{R}(\lambda) = \underset{\boldsymbol{\beta}}{\mathrm{argmin}}\mathrm{PRSS}_{\lambda}(\boldsymbol{\beta}) = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{D}\right)^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{y}.$$

This is a biased estimator of $\boldsymbol{\beta}$ because the conditional expected value is given by

$$E\left[\widehat{\boldsymbol{\beta}}^{R}(\lambda)\right] = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{D}\right)^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta}$$

but as will be described later, relative to the OLS estimator, by introducing a "small" bias, the variance or/and the EPE of this method could potentially be reduced (Wakefield 2013).

By using the method of Lagrange multipliers, the Ridge regression estimates of the $\boldsymbol{\beta}$ coefficients can be reformulated in a similar way to the OLS problem, but subject to the condition that the magnitude of the $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_p)^{\mathrm{T}}$ be less or equal to $t(\lambda)^{\frac{1}{2}}$, that is,

$$\widehat{\boldsymbol{\beta}}^{R}(\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \mathrm{RSS}(\boldsymbol{\beta})$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \leq t(\lambda),$$

where $t(\lambda)$ is a one-to-one function that produces an equivalent definition to the penalized OLS presentation of the Ridge regression described before (Wakefield 2013; Hastie et al. 2009, 2015). This constrained reformulation gives a more transparent role than the one played by the tuning parameter, and among other things, suggests a convenient and common way of redefining the Ridge estimator by standardizing the variables when these are of very different scales.

A graphic representation of this constraint problem for $\beta_0 = 0$ and $p = 2$ is given in Fig. 3.2, where the nested ellipsoids correspond to contour plots of RSS($\boldsymbol{\beta}$) and the green region is the restriction with $t(\lambda) = 3^2$, which contains the Ridge solution.

The MLR defined in (3.1) but now defined with the standardized variables is expressed as

$$\begin{aligned} y &= 1_n\mu + \boldsymbol{X}_{1s}\boldsymbol{\beta}_{0s} + \epsilon \\ &= \boldsymbol{X}_s\boldsymbol{\beta}_s + \epsilon, \end{aligned}$$

where $\mathbf{1}_n$ is the column vector with 1's in all its entries, $\boldsymbol{X}_{1s} = \begin{bmatrix} x_{11s} & \cdots & x_{1ps} \\ \vdots & \vdots & \vdots \\ x_{n1s} & \cdots & x_{nps} \end{bmatrix}$, $x_{ijs} = \left(x_{ij} - \overline{x}_j\right)/s_j$, $s_j = \sqrt{\sum_{i=1}^{n}\left(x_{ij} - \overline{x}_j\right)^2/n}$, $j = 1, \ldots, p$; $\boldsymbol{X}_s = [\mathbf{1}_n\ \boldsymbol{X}_{1s}]$; $\boldsymbol{\beta}_s = \left(\mu,\ \boldsymbol{\beta}_{0s}^{\mathrm{T}}\right)^{\mathrm{T}}$; and $\boldsymbol{\beta}_{0s} = (\beta_{1s}, \ldots, \beta_{ps})^{\mathrm{T}}$.

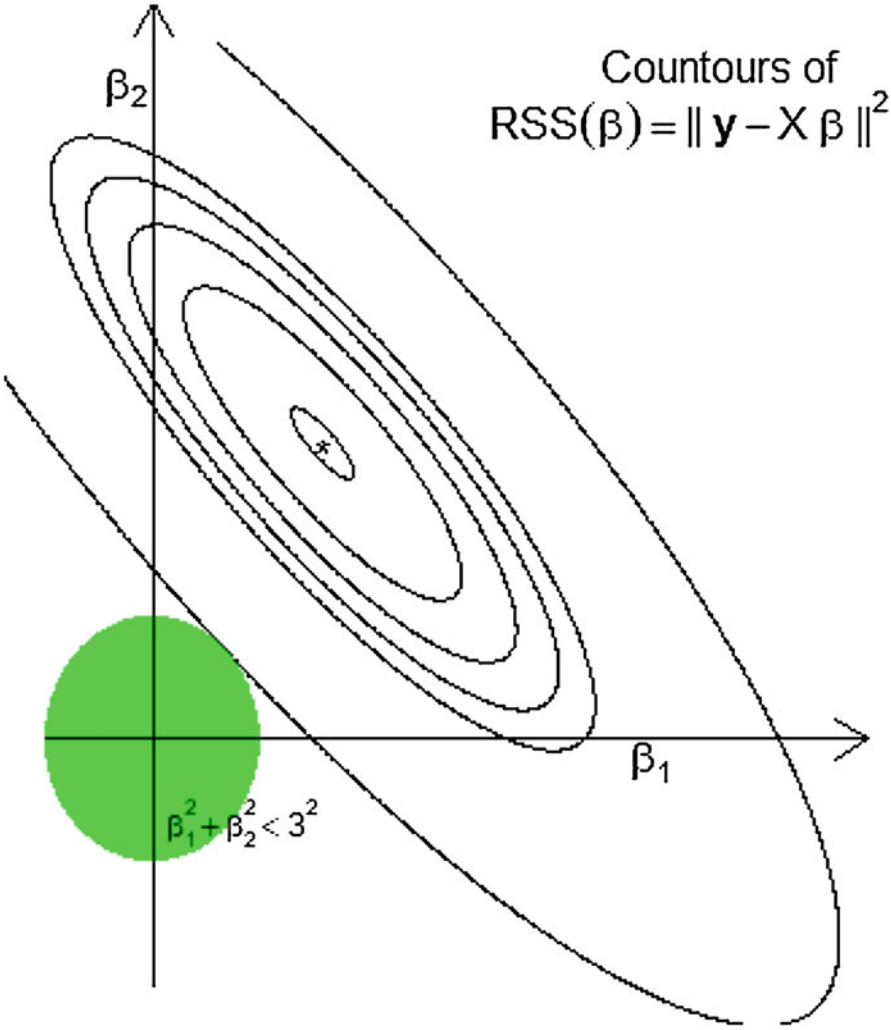

Fig. 3.2 Graphic representation of the Ridge solution of the OLS with restriction $\sum_{j=1}^{p} \beta_j^2 < 3^2$. The green region contains the Ridge solution for $t(\lambda) = 3^2$

Then, the redefined penalized residual sum squared under this model is

$$\begin{aligned} \text{PRSS}_\lambda(\boldsymbol{\beta}_s) &= \sum_{i=1}^{n} \left(y_i - \mu - \sum_{j=1}^{p} x_{ijs} \beta_{js} \right)^2 + \lambda \sum_{j=1}^{p} \beta_{js}^2 \\ &= \left(\boldsymbol{y} - \boldsymbol{X}_s^{\mathrm{T}} \boldsymbol{\beta}_s \right)^{\mathrm{T}} \left(\boldsymbol{y} - \boldsymbol{X}_s^{\mathrm{T}} \boldsymbol{\beta}_s \right) + \lambda \boldsymbol{\beta}_{0s}{}^{\mathrm{T}} \boldsymbol{D} \boldsymbol{\beta}_{0s}. \end{aligned}$$

The Ridge solution under this redefinition is like the one given before, but now

$$\begin{aligned}\widehat{\boldsymbol{\beta}}_s^R(\lambda) &= \left(\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s + \boldsymbol{\lambda D}\right)^{-1}\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{y} \\ &= \left(\begin{bmatrix} 1_n^{\mathrm{T}} \\ \boldsymbol{X}_{1s}^{\mathrm{T}} \end{bmatrix}[1_n\ \boldsymbol{X}_{1s}] + \lambda \boldsymbol{D}\right)^{-1}\begin{bmatrix} 1_n^{\mathrm{T}} \\ \boldsymbol{X}_{1s}^{\mathrm{T}} \end{bmatrix}\boldsymbol{y} \\ &= \begin{bmatrix} n & 0_n^{\mathrm{T}} \\ 0_n & \boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I_p} \end{bmatrix}^{-1}\begin{bmatrix} 1_n^{\mathrm{T}} y \\ \boldsymbol{X}_{1s}^{\mathrm{T}} y \end{bmatrix} \\ &= \begin{bmatrix} \overline{y}_n \\ \widehat{\boldsymbol{\beta}}_{0s}(\lambda) \end{bmatrix},\end{aligned}$$

where $\overline{y}_n = \sum_{i=1}^{n} y_i/n$ is the sample mean of the responses and $\widehat{\boldsymbol{\beta}}_{0s}(\lambda) = \left(\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I_p}\right)^{-1}\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{y}$ is the Ridge estimator of $\boldsymbol{\beta}_{0s}$. The mean value of this Ridge solution is

$$\mathrm{E}\left[\widehat{\boldsymbol{\beta}}_s^R(\lambda)\right] = \begin{bmatrix} \mu \\ E\left[\widehat{\boldsymbol{\beta}}_{0s}(\lambda)\right] \end{bmatrix},$$

where $E\left[\widehat{\boldsymbol{\beta}}_{0s}(\lambda)\right] = \left(\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I_p}\right)^{-1}\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s}\boldsymbol{\beta}_{0s}$ is the expected value of the Ridge estimator of $\boldsymbol{\beta}_{0s}$. The variance–covariance matrix is

$$\begin{aligned}\mathrm{Var}\left(\widehat{\boldsymbol{\beta}}_s^R(\lambda)\right) &= \left(\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s + \boldsymbol{\lambda D}\right)^{-1}\boldsymbol{X}_s^{\mathrm{T}}\mathrm{Var}(\boldsymbol{y})\boldsymbol{X}_s\left(\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s + \boldsymbol{\lambda D}\right)^{-1} \\ &= \sigma^2\begin{bmatrix} n & 0_n^{\mathrm{T}} \\ 0_n & \boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I_p} \end{bmatrix}^{-1}\begin{bmatrix} 1_n^{\mathrm{T}} \\ \boldsymbol{X}_{1s}^{\mathrm{T}} \end{bmatrix}[1_n\ \boldsymbol{X}_{1s}]\begin{bmatrix} n & 0_n^{\mathrm{T}} \\ 0_n & \boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I_p} \end{bmatrix}^{-1} \\ &= \sigma^2\begin{bmatrix} 1/n & 0_n^{\mathrm{T}} \\ 0_n & \mathrm{Var}\left(\widehat{\boldsymbol{\beta}}_{0s}(\lambda)\right) \end{bmatrix},\end{aligned}$$

where $\mathrm{Var}\left(\widehat{\boldsymbol{\beta}}_{0s}(\lambda)\right) = \left(\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I}_p\right)^{-1}\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s}\left(\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} + \lambda \boldsymbol{I}_p\right)^{-1}$. So, because in this standardized way, the Ridge solution of the intercept (μ) is the sample mean of the observed responses, and the correlation of this with the rest of the estimated parameters ($\widehat{\boldsymbol{\beta}}_{0s}(\lambda)$) is null, in the literature it is common to handle this parameter separately from all other coefficients ($\boldsymbol{\beta}_{0s}$) (Christensen 2011).

Note that

$$E\left[\widehat{\boldsymbol{\beta}}_{0s}(\lambda)\right] = \boldsymbol{\Gamma}_s\left(\boldsymbol{\Lambda}_s + \lambda \boldsymbol{I_p}\right)^{-1}\boldsymbol{\Lambda}_s\boldsymbol{\beta}_{0s}^*$$

and

$$\mathrm{Var}\left(\widehat{\boldsymbol{\beta}}_{0s}(\lambda)\right) = \boldsymbol{\Gamma}_s\left(\boldsymbol{\Lambda}_p + \lambda \boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Lambda}_s\left(\boldsymbol{\Lambda}_p + \lambda \boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Gamma}_s^{\mathrm{T}},$$

where $\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} = \boldsymbol{\Gamma}_s\boldsymbol{\Lambda}_s\boldsymbol{\Gamma}_s^{\mathrm{T}}$ is the spectral decomposition of $\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s}$ and $\boldsymbol{\beta}_{0s}^{*} = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s}$. So the conditional expected prediction error at $\boldsymbol{x}_o$ when using the Ridge solution is

$$\begin{aligned}
\mathrm{EPE}_{\lambda}(\boldsymbol{x}_o) &= E_{Y,Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(Y_o - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^{R}(\lambda)\right)^2\right] \\
&= E_{Y,Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(Y_o - E(Y_0|\boldsymbol{x}_0) + E(Y_0|\boldsymbol{x}_0) - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^{R}(\lambda)\right)^2\right] \\
&= \sigma^2 + E_{Y,Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\beta}_s - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^{R}(\lambda)\right)^2\right] \\
&= \sigma^2 + \left[\left(\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\beta}_s - \boldsymbol{x}_o^{*\mathrm{T}}E_{Y|X}\left(\widehat{\boldsymbol{\beta}}_s^{R}(\lambda)\right)\right)^2\right] + \mathrm{Var}\left(\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^{R}(\lambda)|\boldsymbol{x}_{\boldsymbol{o}}\right) \\
&= \sigma^2 + \left[\left(\mu + \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\beta}_{0s} - \mu - \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_s\left(\boldsymbol{\Lambda}_s + \lambda\boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Lambda}_s\boldsymbol{\beta}_{0s}^{*}\right)^2\right] + \sigma^2\left[\frac{1}{n} + \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_s\left(\boldsymbol{\Lambda}_p + \lambda\boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Lambda}_s\left(\boldsymbol{\Lambda}_p + \lambda\boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Gamma}_s^{T}\boldsymbol{x}_o\right] \\
&= \sigma^2 + \left[\left(\boldsymbol{x}_o^{**\mathrm{T}}\boldsymbol{\beta}_{0s}^{*} - \boldsymbol{x}_o^{**\mathrm{T}}\left(\boldsymbol{\Lambda}_s + \lambda\boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Lambda}_s\boldsymbol{\beta}_{0s}^{*}\right)^2\right] + \sigma^2\left[\frac{1}{n} + \boldsymbol{x}_o^{**\mathrm{T}}\left(\boldsymbol{\Lambda}_p + \lambda\boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{\Lambda}_s\left(\boldsymbol{\Lambda}_p + \lambda\boldsymbol{I}_{\boldsymbol{p}}\right)^{-1}\boldsymbol{x}_o^{**}\right] \\
&= \sigma^2 + \left[\sum_{j=1}^{p}\left(1 - \frac{\lambda_j}{\lambda_j + \lambda}\right)x_{oj}^{**}\beta_{js}^{*}\right]^2 + \sigma^2\left(\frac{1}{n} + \sum_{j=1}^{p}\frac{\lambda_j}{\left(\lambda_j + \lambda\right)^2}\left(x_{oj}^{**}\right)^2\right),
\end{aligned}$$

where $\boldsymbol{x}_o^{**} = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{x}_o^{*} = \left(x_{o0}^{**}, \ldots, x_{op}^{**}\right)^{\mathrm{T}}$ and $\boldsymbol{\beta}_{0s}^{*} = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s} = \left(\beta_{1s}^{*}, \ldots, \beta_{ps}^{*}\right)^{\mathrm{T}}$. The second and third terms of the last equality correspond to the squared bias and the variance of $\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^{R}(\lambda)$ as an estimator of $\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\beta}_s$, respectively. By setting $\lambda = 0$, this EPE corresponds to the EPE of the OLS prediction but with standardized variables, while by letting λ be very large, the variance will decrease and the squared bias will increase.

More importantly, because the derivative of $\mathrm{EPE}_{\lambda}(\boldsymbol{x}_o)$ with respect to λ, $\frac{d}{d\lambda}\mathrm{PE}_{\lambda}(\boldsymbol{x}_o)$, is a right continuous function at $\lambda = 0$, and for $\boldsymbol{X}_{1s}$ of full column rank, $\lim_{\lambda\to 0^{+}}\frac{d}{d\lambda}\mathrm{PE}_{\lambda}(\boldsymbol{x}_o) = -2\sigma^2\sum_{j=1}^{p}\frac{\left(x_{oj}^{**}\right)^2}{\lambda_j^2} = c$; then for $\epsilon = -\frac{c}{2} > 0$, we have that $\lambda^{*} > 0$ such that $\left|\frac{d}{d\lambda}\mathrm{PE}_{\lambda}(\boldsymbol{x}_o) - c\right| < \epsilon$ for $\lambda < \lambda^{*}$. From this we have that $\frac{d}{d\lambda}\mathrm{PE}_{\lambda}(\boldsymbol{x}_o) < -\frac{c}{2} + c = \frac{c}{2} < 0$ for all $\lambda < \lambda^{*}$, for some $\lambda^{*} > 0$. Then, at least in the interval $[0, \lambda^{*}]$, the expected prediction error at $\boldsymbol{x}_o$ shows a decreasing behavior, which indicates that there is a value of λ such that with the Ridge regression estimation of beta coefficients, we can get a smaller prediction error than with the OLS prediction. Figure 3.3 shows a graphic representation of this behavior of Ridge prediction, where the lower EPE is reached at about $\lambda =$ exp (2.22). Figure 3.3 also shows the increasing and decreasing behavior of the bias-squared and the variance involved.

When $\boldsymbol{X}_{1s}$ is not full column rank, the previous argument regarding the behavior of the EPE of the Ridge solution is already not valid directly, but it could be used for

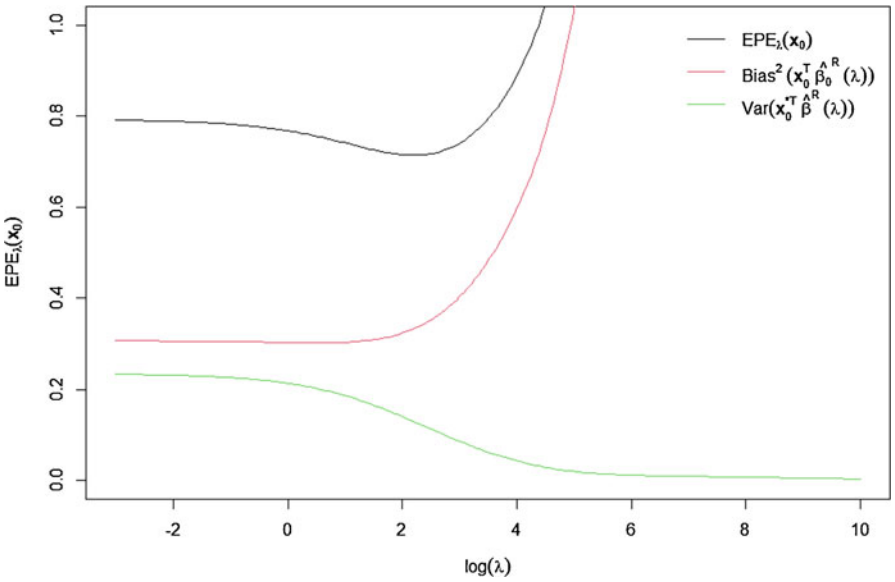

Fig. 3.3 Behavior of the expected prediction error at $\boldsymbol{x}_o$ of the Ridge solution

validating part of the more general case. To see this, first note that the spectral decomposition of $\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s}$ can be reduced to

$$\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s} = [\boldsymbol{\Gamma}_{1s}\ \boldsymbol{\Gamma}_{2s}]\begin{bmatrix}\boldsymbol{\Lambda}_{1s} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \boldsymbol{\Lambda}_{2s}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Gamma}_{1s}^{\mathrm{T}} \\ \boldsymbol{\Gamma}_{2s}^{\mathrm{T}}\end{bmatrix} = \boldsymbol{\Gamma}_{1s}^{\mathrm{T}}\boldsymbol{\Lambda}_{1s}\boldsymbol{\Gamma}_{1s},$$

where $\boldsymbol{\Lambda}_{1s} = \mathrm{Diag}\left(\lambda_1, \ldots, \lambda_{p^*}\right)$, $p^* = \mathrm{rank}\,(\mathbf{X}_{1c})$ is the rank of design matrix and $\boldsymbol{\Lambda}_{2s}$ is the null matrix of order $(p - p^*) \times (p - p^*)$. Furthermore, because $\boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\Gamma}_s = \boldsymbol{I}_p$ implies that $\boldsymbol{\Gamma}_{2s}^{\mathrm{T}}\boldsymbol{X}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1s}\boldsymbol{\Gamma}_{2s} = \mathbf{0}$, which in turn implies that $\boldsymbol{X}_{1s}\boldsymbol{\Gamma}_{2s} = \mathbf{0}$, then the MLR can be conveniently expressed by

$$\begin{aligned}\boldsymbol{y} &= 1_n\mu + \boldsymbol{X}_{1s}\boldsymbol{\beta}_{0s} + \epsilon \\ &= 1_n\mu + \boldsymbol{X}_{1s}\boldsymbol{\Gamma}_s\boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s} + \epsilon \\ &= 1_n\mu + \boldsymbol{X}_{1s}[\boldsymbol{\Gamma}_{1s}\ \boldsymbol{\Gamma}_{2s}]\boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s} + \epsilon \\ &= 1_n\mu + [\boldsymbol{X}_{1s}\boldsymbol{\Gamma}_{1s}\ \boldsymbol{X}_{1s}\boldsymbol{\Gamma}_{2s}]\boldsymbol{\beta}_{0s}^* + \epsilon \\ &= 1_n\mu + \boldsymbol{X}_{1s}^*\boldsymbol{\beta}_{01s}^* + \epsilon,\end{aligned}$$

where $\boldsymbol{\beta}_{0s}^* = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s}$, $\boldsymbol{X}_{1s}^* = \boldsymbol{X}_{1s}\boldsymbol{\Gamma}_{1s}$, and $\boldsymbol{\beta}_{0s}^* = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s} = \left[\boldsymbol{\beta}_{0s}^T\boldsymbol{\Gamma}_{1s}, \boldsymbol{\beta}_{0s}^{\mathrm{T}}\boldsymbol{\Gamma}_{2s}\right]^{\mathrm{T}} = \left[\boldsymbol{\beta}_{01s}^{*\,\mathrm{T}}\ \boldsymbol{\beta}_{02s}^{*\mathrm{T}}\right]^{\mathrm{T}}$.

Also, from similar arguments, note that the penalized residual sum of squares of the Ridge solution can be expressed by

$$
\begin{aligned}
&(\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_{1s}\boldsymbol{\beta}_{0s})^{\mathrm{T}}(\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_{1s}\boldsymbol{\beta}_{0s}) + \lambda\boldsymbol{\beta}_{0s}^{\mathrm{T}}\boldsymbol{\beta}_{0s} \\
&= \left(\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_{1s}^*\boldsymbol{\beta}_{01s}^*\right)^{\mathrm{T}}\left(\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_{1s}^*\boldsymbol{\beta}_{01s}^*\right) + \lambda\boldsymbol{\beta}_{0s}^{\mathrm{T}}\left(\boldsymbol{\Gamma}_{1s}^{\mathrm{T}}\boldsymbol{\Gamma}_{1s} + \boldsymbol{\Gamma}_{2s}^{\mathrm{T}}\boldsymbol{\Gamma}_{2s}\right)\boldsymbol{\beta}_{0s} \\
&= \left(\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_{1s}^*\boldsymbol{\beta}_{01s}^*\right)^{\mathrm{T}}\left(\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_{1s}^*\boldsymbol{\beta}_{01s}^*\right) + \lambda\boldsymbol{\beta}_{01s}^{*\mathrm{T}}\boldsymbol{\beta}_{01s}^* + \lambda\boldsymbol{\beta}_{02s}^{*\mathrm{T}}\boldsymbol{\beta}_{02s}^*
\end{aligned}
$$

This function of $\boldsymbol{\beta}_{0s}^*$ is minimized at $\widetilde{\boldsymbol{\beta}}_{0s}^*(\lambda) = \left(\widetilde{\boldsymbol{\beta}}_{01s}^{*\mathrm{T}}(\lambda), \boldsymbol{0}_{p-p^*}^{\mathrm{T}}\right)^{\mathrm{T}}$, where $\widetilde{\boldsymbol{\beta}}_{01s}^{*\mathrm{T}}(\lambda) = \left(\boldsymbol{\Lambda}_{1s} + \lambda\boldsymbol{I}_{p^*}\right)^{-1}\boldsymbol{\Gamma}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1S}^{\mathrm{T}}\boldsymbol{y}$ is the Ridge solution of the MLR expressed in terms of $\boldsymbol{X}_{1s}^*\boldsymbol{\beta}_{0s}^*$. Furthermore, because $\boldsymbol{\beta}_{0s}^* = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s}$ is a non-singular transformation, the original Ridge solution of $\boldsymbol{\beta}_s$ can be expressed in terms of $\widetilde{\boldsymbol{\beta}}_{0s}^*(\lambda)$ as $\widehat{\boldsymbol{\beta}}_s(\lambda) = \left(\overline{y}_n, \widehat{\boldsymbol{\beta}}_{0s}^{*\mathrm{T}}(\lambda)\right)^{\mathrm{T}}$, where $\widehat{\boldsymbol{\beta}}_{0s}(\lambda) = \boldsymbol{\Gamma}_s\widehat{\boldsymbol{\beta}}_{0s}^{*\mathrm{T}}(\lambda) = \boldsymbol{\Gamma}_{1s}\widetilde{\boldsymbol{\beta}}_{01s}^*(\lambda)$. Then, in a similar fashion as before, the conditional expected prediction error at $\boldsymbol{x}_o$ by using the Ridge solution in this case can be computed as

$$
\begin{aligned}
&\mathrm{EPE}_\lambda(\boldsymbol{x}_o) \\
&= \sigma^2 + E_{\boldsymbol{Y},Y_o|\boldsymbol{X},\boldsymbol{x}_o}\left[\left(\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\beta}_s - \boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^R(\lambda)\right)^2\right] \\
&= \sigma^2 + \left[\left(\boldsymbol{x}_o^{*\mathrm{T}}\boldsymbol{\beta}_s - \boldsymbol{x}_o^{*\mathrm{T}}E_{Y|X}\left(\widehat{\boldsymbol{\beta}}_s^R(\lambda)\right)\right)^2\right] + \mathrm{Var}\left(\boldsymbol{x}_o^{*\mathrm{T}}\widehat{\boldsymbol{\beta}}_s^R(\lambda)|\boldsymbol{x}_o\right) \\
&= \sigma^2 + \left[\left(\mu + \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_s\boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s} - \mu - \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_{1s}E_{Y|X}\left(\widehat{\boldsymbol{\beta}}_{0s}^{*\mathrm{T}}(\lambda)\right)\right)^2\right] + \left[\frac{\sigma^2}{n} + \mathrm{Var}\left(\boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_s\widehat{\boldsymbol{\beta}}_{0s}^{*\mathrm{T}}(\lambda)|\boldsymbol{x}_o\right)\right] \\
&= \sigma^2 + \left[\left(\mu + \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_s\boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{0s} - \mu - \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_{1s}\left(\boldsymbol{\Lambda}_{1s} + \lambda\boldsymbol{I}_{p^*}\right)^{-1}\boldsymbol{\Gamma}_{1s}^{\mathrm{T}}\boldsymbol{X}_{1c}^{\mathrm{T}}\boldsymbol{X}_{1c}\boldsymbol{\beta}_{0S}\right)^2\right] \\
&\quad + \sigma^2\left[\frac{1}{n} + \boldsymbol{x}_o^{\mathrm{T}}\boldsymbol{\Gamma}_{1s}\left(\boldsymbol{\Lambda}_{1s} + \lambda\boldsymbol{I}_{p^*}\right)^{-1}\boldsymbol{\Lambda}_{1s}\left(\boldsymbol{\Lambda}_{1s} + \lambda\boldsymbol{I}_{p^*}\right)^{-1}\boldsymbol{\Gamma}_{1s}^{T}\boldsymbol{x}_o\right] \\
&= \begin{cases}
\sigma^2 + \left[\left(\sum_{j=1}^{p^*}\left(1 - \frac{\lambda_j}{\lambda_j + \lambda}\right)x_{oj}^*\beta_{oj}^*\right)^2\right] + \sigma^2\left[\frac{1}{n} + \sum_{j=1}^{p^*}\frac{\lambda_j}{\left(\lambda_j + \lambda\right)^2}\left(x_{oj}^*\right)^2\right] & \text{if } x_o = \boldsymbol{\Gamma}_{1s}\boldsymbol{a}_1 \\
\sigma^2 + \left[\boldsymbol{x}_o^T\boldsymbol{\beta}_{0s}\right]^2 + \frac{\sigma^2}{n} & \text{if } x_o = \boldsymbol{\Gamma}_{2s}\boldsymbol{a}_2,
\end{cases}
\end{aligned}
$$

where $\boldsymbol{x}_o^* = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{x}_o = \left[x_{o1}^*, \ldots, x_{op}^*\right]^{\mathrm{T}}$, $\boldsymbol{\beta}_o^* = \boldsymbol{\Gamma}_s^{\mathrm{T}}\boldsymbol{\beta}_{os} = \left[\beta_{o1}^*, \ldots, \beta_{op}^*\right]^{\mathrm{T}}$, and $\boldsymbol{a}_1 \in \mathbb{R}^{p^*}$ and $\boldsymbol{a}_2 \in \mathbb{R}^{p-p^*}$. So, using a similar argument as before, in the first case ($x_o = \boldsymbol{\Gamma}_{1s}\boldsymbol{a}_1$), the value of $\lambda > 0$ is such that the expected prediction error at $\boldsymbol{x}_o$ is better than that obtained with the OLS approach, $\widehat{\boldsymbol{\beta}}_s(0) = \lim_{\lambda\to 0}\widehat{\boldsymbol{\beta}}_s(\lambda) = \left[\overline{y}_n, \left(\boldsymbol{\Gamma}_{1s}\Lambda_{1s}^{-1}\boldsymbol{\Gamma}_{1s}^{\mathrm{T}}X_{1c}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\right]^{\mathrm{T}}$. In the second case, $x_o = \boldsymbol{\Gamma}_{2s}\boldsymbol{a}_2$, the EPE($\boldsymbol{x}_o$) in both approaches is the same and doesn't depend on λ, so in such cases, no improved gain with regard to the Ridge solution is achieved. A third case was included, that is, when the target feature is of the form $\boldsymbol{x}_o = \boldsymbol{\Gamma}_{1s}\boldsymbol{a}_1 + \boldsymbol{\Gamma}_{2s}\boldsymbol{a}_2$. In this case, under the described argument, the advantage of Ridge regression over the OLS approach in a prediction context is not clear.

However, in practice, we don't know the true value of the parameters, and we need to evaluate the test error in all possible values of the training sample, which we also don't have. So a common way to choose the λ value is by cross-validation. For more details about validation strategies, see Chap. 4. For example, with a k-fold CV, the complete data set is divided into K balanced disjoint subsets, S_k, $k = 1, \ldots, K$. One subset is used as validation and the rest are used to fit the model in each value of a chosen grid of values of λ. This procedure is repeated K times, where each time a subset in the partition is taken as the validation set. A more detailed k-fold CV procedure is described below:

1. First, choose a grid of values of λ, $\lambda = (\lambda_1, \ldots, \lambda_L)$.
2. Remove the subset S_k and for each value λ_l in the grid, fit the model with the remaining $K - 1$ elements of the partition denoted by $\widehat{\boldsymbol{\beta}}^R_{-k}(\lambda_l)$, the corresponding Ridge estimation of $\boldsymbol{\beta}$, and compute the average prediction error across all observations in the validation set S_k as

$$\widehat{\text{APE}}_{-k}(\lambda_l) = \frac{1}{|S_k|} \sum_{y_i \in S_k} \left(y_i - \boldsymbol{x}_i^{*\text{T}} \widehat{\boldsymbol{\beta}}^R_{-k}(\lambda_l) \right)^2,$$

   where $|S_k|$ denotes the total observations in partition k.
3. Choose as the best value of λ in the grid $\left(\widetilde{\lambda}^*\right)$, the one with the lower average prediction error across all partitions, that is

$$\widetilde{\lambda}^* = \arg\min_{\lambda_l} \widehat{\text{APE}}(\lambda_l),$$

   where $\widehat{\text{APE}}(\lambda_l) = \frac{1}{K} \sum_{k=1}^{K} \widehat{\text{APE}}_{-k}(\lambda_l)$.
4. Once $\widetilde{\lambda}^*$ is chosen, we fit the model with the complete data set and the prediction of new individuals with feature $\boldsymbol{x}_o$ can be made with $\widehat{y}_i = \boldsymbol{x}_o^{*T} \widehat{\boldsymbol{\beta}}^R \left(\widetilde{\lambda}^*\right)$, where $\widehat{\boldsymbol{\beta}}^R \left(\widetilde{\lambda}^*\right)$ is the Ridge estimation of $\boldsymbol{\beta}$ at $\lambda = \widetilde{\lambda}^*$.

It is important to point out that very often the performance of the model needs to be evaluated for comparison purposes with other competing models. A common way to do this is to split the data set several times into two subsets, one for training the model (D_{tr}) (to fit the model) and the other for testing (D_{tst}) it, in which the predictive ability of a model is tested. In each splitting, only the training data set (D_{tr}) is used to train the model (by steps 1–3 before fitting the whole training data set), and the prediction evaluation of the fitted model is made with the testing data set, as explained before in point 4. The prediction evaluation of the testing data set is done by an empirical "estimate" of the EPE, $\text{MSE} = \frac{1}{|D_{\text{tst}}|} \sum_{i \in D_{\text{tst}}} \left(y_i - \boldsymbol{x}_o^{*\text{T}} \widehat{\boldsymbol{\beta}}^R \left(\widetilde{\lambda}^*\right), \right)^2$, and

finally, an average evaluation of the performance of the model is obtained across all chosen splittings. See Chap. 4 for more explicit details.

The Ridge solution can also be obtained from a Bayesian formulation. To do this, consider the MLR model described before with standardized features and the vector of residuals distributed as $N_n(\mathbf{0}, \sigma^2 \boldsymbol{I}_n)$. With this assumption, the vector of responses $\boldsymbol{y}$ is distributed in a multivariate normal distribution with vector mean $\mathbf{1}_n\mu + \boldsymbol{X}_{1s}\boldsymbol{\beta}_{0s}$ and variance–covariance matrix $\sigma^2\boldsymbol{I}_n$. Then, to complete the Bayesian formulation, assume $\boldsymbol{\beta}_{os} \sim N_p\left(\mathbf{0}, \sigma_\beta^2 \boldsymbol{I}_p\right)$ as the prior distribution of the beta coefficients in $\boldsymbol{\beta}_{0s}$ and a "flat" prior for the intercept μ, where σ^2 and σ_β^2 are known. Under this Bayesian specification, the posterior distribution of $\boldsymbol{\beta}_s$ is

$$
\begin{aligned}
& f(\boldsymbol{\beta}_s|\boldsymbol{y}, \boldsymbol{X}_{1s}) \\
& \propto \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}_s\boldsymbol{\beta}_s)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}_s\boldsymbol{\beta}_s)\right] \exp\left(-\frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}_{0s}^{\mathrm{T}}\boldsymbol{\beta}_{0s}\right) \\
& \propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{\beta}_s^{\mathrm{T}}\left(\sigma_\beta^{-2}\boldsymbol{D} + \sigma^{-2}\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s\right)\boldsymbol{\beta}_s - 2\sigma^{-2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}_s\boldsymbol{\beta}_s\right]\right\} \\
& \propto \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}_s - \widetilde{\boldsymbol{\beta}}_s\right)^{\mathrm{T}}\widetilde{\boldsymbol{\Sigma}}_\beta^{-1}\left(\boldsymbol{\beta}_s - \widetilde{\boldsymbol{\beta}}_s\right)\right\},
\end{aligned}
$$

where $\widetilde{\boldsymbol{\Sigma}}_\beta = \left(\sigma_\beta^{-2}\boldsymbol{D} + \sigma^{-2}\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s\right)^{-1} = \sigma^2\left(\sigma^2/\sigma_\beta^2\boldsymbol{D} + \boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s\right)^{-1}$, $\widetilde{\boldsymbol{\beta}}_s = \sigma^{-2}\widetilde{\boldsymbol{\Sigma}}_\beta\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{y}$, and $\boldsymbol{D}$ is the diagonal penalty matrix. That is, the posterior distribution of $\boldsymbol{\beta}_s$ is a multivariate normal distribution with vector mean $\widetilde{\boldsymbol{\beta}}_s = \sigma^{-2}\widetilde{\boldsymbol{\Sigma}}_\beta\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{y} = \left(\sigma^2/\sigma_\beta^2\boldsymbol{D} + \boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s\right)^{-1}\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{y}$ and variance–covariance matrix $\widetilde{\boldsymbol{\Sigma}}_\beta = \sigma^2\left(\sigma^2/\sigma_\beta^2\boldsymbol{D} + \boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{X}_s\right)^{-1}$. Then, by taking $\lambda = \sigma^2/\sigma_\beta^2$, we have that the mean/mode of the posterior distribution of $\boldsymbol{\beta}_s$ coincides with the Ridge estimation described before, $\widetilde{\boldsymbol{\beta}}^R(\lambda)$.

Example 2 We considered a genomic example to illustrate the Ridge regression approach and the CV process to choose the learning parameter λ (WheatMadaToy, PH the response). This data set consists of 50 observations corresponding to 50 lines and a relationship genomic matrix computed from marker information. Table 3.1 shows the prediction behavior of the Ridge and the OLS approaches in terms of the MSEP, across five different splittings obtained by partitioning the complete data set into five subsets: the data of a subset are used as a testing set and the rest to train the model. For training the model, a five-fold cross-validation (5FCV) was used along the lines following steps 1–3 described before, and the prediction performance was done following step 4.

Table 3.1 Prediction behavior of the Ridge and OLS regression models across different partitions of the complete data set: one subset of the partition (20%) is used for evaluating the performance of the model and the rest (80%) for training the model. RR denotes Ridge regression method

Partition	MSE RR	MSE OLS
1	325.40	379.33
2	433.19	454.37
3	803.76	1341.35
4	319.53	312.81
5	403.62	555.10
Average	457.10	608.59

Table 3.1 indicates that in four out of five partitions, the Ridge regression shows less MSE than the corresponding OLS approach. In all these cases, the MSE of the OLS was, on average, 31.46% greater than that of the Ridge regression approach, and in general, on average, by 31.14% (MSE = 421.8834 for Ridge and MSE = 655.8596 for OLS). From this, we have that the Ridge regression approach shows a better prediction performance than the OLS. The large variation of the MSE between folds observed in this example could indicate that for obtaining a more precise comparison between models, a larger number of partitions need to be used. Often, the use of more partitions is avoided when larger data sets are used in applications.

The R code used for obtaining this result is the following:

```
######################R code for Example 2 ############################
rm(list=ls())
library(BMTME)
data("WheatMadaToy")
dat_F = phenoMada
dim(dat_F)
dat_F$GID = as.character(dat_F$GID)
G = genoMada
eig_G = eigen(G)
G_0.5 = eig_G$vectors%*%diag(sqrt(eig_G$values))%*%t(eig_G$vectors)
X = G_0.5
y = dat_F$PH
n = length(y)
source('TR_RR.R')
#5FCV
set.seed(3)
K = 5
Tab = data.frame()
Grpv = findInterval(cut(sample(1:n,n),breaks=K),1:n)
for(i in 1:K)
{
 Pos_tr = which(Grpv!=i)
 y_tr = y[Pos_tr]
 X_tr = X[Pos_tr,]
 TR_RR = Tr_RR_f(y_tr,X_tr,K=5,KG=100,KR=1)
 lambv = TR_RR$lambv
```

```
 #Tst
 y_tst = y[-Pos_tr]; X_tst = X[-Pos_tr,]
 #RR
 Pred_RR = Pred_RR_f(y_tst,X_tst,TR_RR)
 #OLS
 Pred_ols = Pred_ols_f(y_tst,X_tst,y_tr,X_tr)
 Tab = rbind(Tab,data.frame(Sim=i,MSEP_RR = Pred_RR$MSEP,
                MSEP_ols = Pred_ols$MSEP))
 cat('i = ', i,'\n')
}
Tab
```

Tr_RR_f, Pred_RR_f, and Pred_ols_f are R functions accessed by the command source('TR_RR.R'), where TR_RR.R is the file R script defined in Appendix 1.

From the last code, three things are important to point out:

1. The Grpv contains the information of the K=5 folds for the outer CV implemented and each time the model is trained with $K - 1$ and tested with the remaining fold.
2. The function that trains the model under Ridge regression is called Tr_RR_f, while the function that obtains the predictions of the testing set of this trained model is Pred_RR_f; both functions are fully described in Appendix 1.
3. The predictions under the OLS method are obtained with the function Pred_ols_f, which is also fully detailed in Appendix 1. It is important to point out that the function Tr_RR_f internally implements an inner k-fold CV to tune the hyperparameter λ required in Ridge regression.

Example 3 (Simulation) To get a better idea about the behavior of the Ridge solution, here we report the results of a small simulation study in a scenario where the number of observations (n=100) is less than the number of features ($p = 500$) and these are moderately correlated. Specifically, we generated 100 data sets, each of size 100, from the following model:

$$y_i = 5 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0 + \epsilon_i,$$

where the vector of beta coefficients ($\boldsymbol{\beta}_0$) was set to the values shown in Fig. 3.4, and the features of all the individuals in each data set were generated from a multivariate normal distribution centered on the null vector and variance–covariance matrix $\boldsymbol{\Sigma} = 0.25\boldsymbol{I}_p + 0.75\boldsymbol{J}_p$, where $\boldsymbol{I_p}$ and $\boldsymbol{J}_p$ are the identity matrix and matrix of ones of dimension $p \times p$. The random errors (ϵ_i) were simulated from a normal distribution with mean 0 and variance 0.025.

The behavior of the Ridge and OLS solutions across the 100 simulated data sets is shown in Fig. 3.5. The MSE of Ridge regression is located on the x-axis and the corresponding MSE of the OLS is located on the y-axis. On average, the OLS resulted in an MSE equal to 808.81, which is 30.59% larger than the average MSE

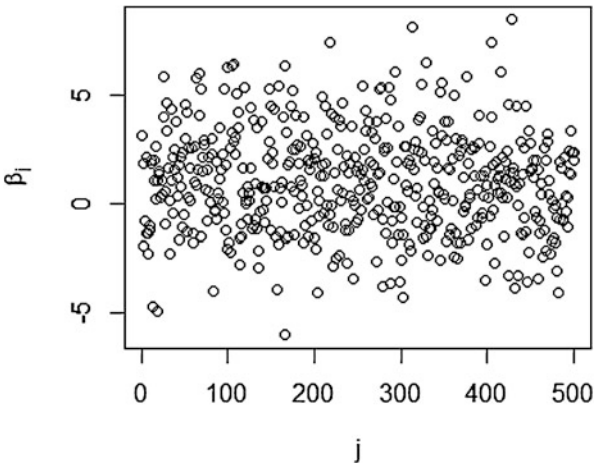

Fig. 3.4 Beta coefficients used in simulation: $\beta_j, j = 1, \ldots, p$

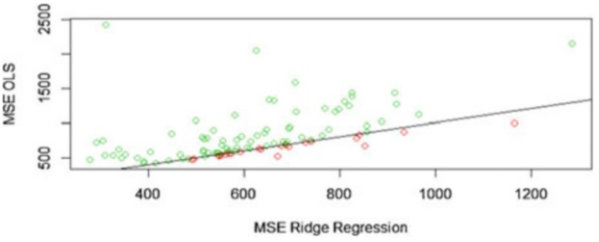

Fig. 3.5 MSE of Ridge regression (MSE RR) versus MSE OLS regression (MSE OLS)

(619.32) of the Ridge approach. In terms of the percentage of simulations in favor of each method, Ridge regression was better in 78 out of 100 simulations, while the OLS was better only in 22 out of 100 simulations. In general, from this small simulation study we obtained more evidence in favor of the Ridge regression method.

The R code used for obtaining this result is the following:

```
############################R code for Example 3######################
rm(list=ls(all=TRUE))
library(mvtnorm)
library(MASS)
source('TR_RR.R')
set.seed(10)
n = 100
p = 500
Var = 0.25*diag(p)+0.75
Tab = data.frame()
betav = rnorm(p,rpois(p,1),2)
plot(betav,xlab=expression(j),ylab=expression(beta[j]))
for(i in 1:100)
{
 X = rmvnorm(n,rep(0,p),Var)
```

```
 dim(X)
 y = 5+X%*%betav + rnorm(n,0,0.5)
 Pos_tr = sample(1:n,n*0.80)
 y_tr = y[Pos_tr]; X_tr = X[Pos_tr,]
 y_tst = y[-Pos_tr]; X_tst = X[-Pos_tr,]
 #Training RR
 TR_RR = Tr_RR_f(y_tr,X_tr,K=5,KG=100,KR=1)
 TR_RR$lamb_o
 lambv = TR_RR$lambv
 plot(log(TR_RR$lambv),TR_RR$iPEv_mean)
 #Prediction RR in testing data
 Pred_RR = Pred_RR_f(y_tst,X_tst,TR_RR)
 Pred_RR$MSEP

 #OLS
 Pred_ols = Pred_ols_f(y_tst,X_tst,y_tr,X_tr)
 Pred_ols

 Tab = rbind(Tab,data.frame(Sim=i,MSEP_RR = Pred_RR$MSEP,
               MSEP_ols = Pred_ols$MSEP))
 cat('i = ', i,'\n')}

Mean_v = colMeans(Tab)
(Mean_v[3]-Mean_v[2])/Mean_v[2]*100

mean(Tab$MSEP_RR<Tab$MSEP_ols)
Pos = which(Tab$MSEP_RR<Tab$MSEP_ols)
mean((Tab$MSEP_ols[Pos]-Tab$MSEP_RR[Pos])/Tab$MSEP_RR[Pos])*100
mean((Tab$MSEP_RR[-Pos]-Tab$MSEP_ols[-Pos])/Tab$MSEP_ols[-Pos])
*100

plot(Tab$MSEP_RR, Tab$MSEP_ols,
   col=ifelse(Tab$MSEP_RR<Tab$MSEP_ols,3,2),
   xlab='MSEP RR', ylab='MSEP OLS')
abline(a=0,b=1)
```

The TR_RR.R script file is the same as the one defined in Example 2 in Appendix 1.

3.6.2 Lasso Regression

Like Ridge regression, the Lasso regression solves the OLS problem but penalizes the residual sum squared in a slightly different way. With the standardized variables, the Lasso estimator of $\boldsymbol{\beta}_s$ is defined as

$$\widetilde{\boldsymbol{\beta}}_s^L(\lambda) = \arg\min_{\mu,\boldsymbol{\beta}_{0s}} \mathrm{PRSS}_\lambda(\boldsymbol{\beta}_s),$$

where now $\text{PRSS}_\lambda(\boldsymbol{\beta}_s) = \sum_{i=1}^{n}\left(y_i - \mu - \sum_{j=1}^{p} x_{ijs}\beta_{js}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_{js}|$ is the $\text{RSS}(\boldsymbol{\beta})$ but penalized by the sum of the absolute regression coefficients. For $\lambda = 0$, the solution is the OLS, while when λ is large, the OLS solutions are shrunken toward 0 (Tibshirani 1996).

Note that for any given values of $\boldsymbol{\beta}_{0s}$, the value of μ that minimizes $\text{PRSS}_\lambda(\boldsymbol{\beta}_s)$ is the sample mean of the responses, $\widetilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} y_i$, the same as the Ridge estimator. However, the rest of the Lasso estimator of $\boldsymbol{\beta}_s$, $\boldsymbol{\beta}_{0s}$, cannot be obtained analytically, so numerical methods are often used.

Although there are efficient algorithms for computing the entire regularization path for the Lasso regression coefficients (Efron et al. 2004; Friedman et al. 2008), here we will describe the coordinate-wise descent given in Friedman et al. (2007). The idea of this method is to successively optimize the $\text{PRSS}_\lambda(\boldsymbol{\beta}_s)$ one parameter at a time (beta coefficient). Holding β_{ks}, $j \neq k$, fixed at their current values $\widetilde{\beta}_{js}(\lambda)$, the value of β_k that minimizes $\text{PRSS}_\lambda(\boldsymbol{\beta}_s)$ is given by

$$\begin{aligned}\widetilde{\beta}^{*}_{ks}(\lambda) &= S\left(\sum_{i=1}^{n} x_{ijs}\left(y_i - \widetilde{y}_i^{(k)}\right), \lambda\right) \\ &= S\left(n\widetilde{\beta}_{ks}(\lambda) + \sum_{i=1}^{n} x_{ijs}(y_i - \widetilde{y}_i), \lambda\right),\end{aligned}$$

where $\widetilde{y}_i^{(k)} = \overline{y} + \sum_{\substack{j=1 \\ j\neq k}}^{p} x_{ijs}\widetilde{\beta}_{js}(\lambda)$ and $S(\beta,\lambda) = \begin{cases} \beta - \lambda & \text{if } \beta > 0 \text{ and } \lambda < |\beta| \\ \beta + \lambda & \text{if } \beta < 0 \text{ and } \lambda < |\beta| \\ 0 & \text{if } \lambda \geq |\beta| \end{cases}$. To obtain the Lasso estimate of $\boldsymbol{\beta}_{0s}$, this process is repeated across all the coefficients until a convergence threshold criterion is reached.

This algorithm can be implemented with the glmnet R package (Friedman et al. 2010) as part of a more general penalty regression (elastic net), which is defined as a combination of the Ridge and Lasso penalties. Due to the structure of the algorithm, this can be used on very large data sets and can benefit from sparsity in the explanatory variables (Friedman et al. 2008).

Equivalently, the Lasso estimator of beta coefficients $\boldsymbol{\beta}_{0s}$ can be defined as

$$\begin{aligned}\widetilde{\boldsymbol{\beta}}^{L}_{0s}(\lambda) &= \underset{\boldsymbol{\beta}_{0s}}{\text{argmin}} \sum_{i=1}^{n}\left(y_i - \overline{y} - \sum_{j=1}^{p} x_{ijs}\beta_{js}\right)^2 \\ \text{subject to } &\sum_{j=1}^{p}|\beta_{js}| \leq t\end{aligned}$$

With this, a graphic representation of the Lasso estimator is like the Ridge (see Fig. 3.6). The nested ellipsoids correspond to contour plots of $\text{RSS}(\boldsymbol{\beta})$ and the green

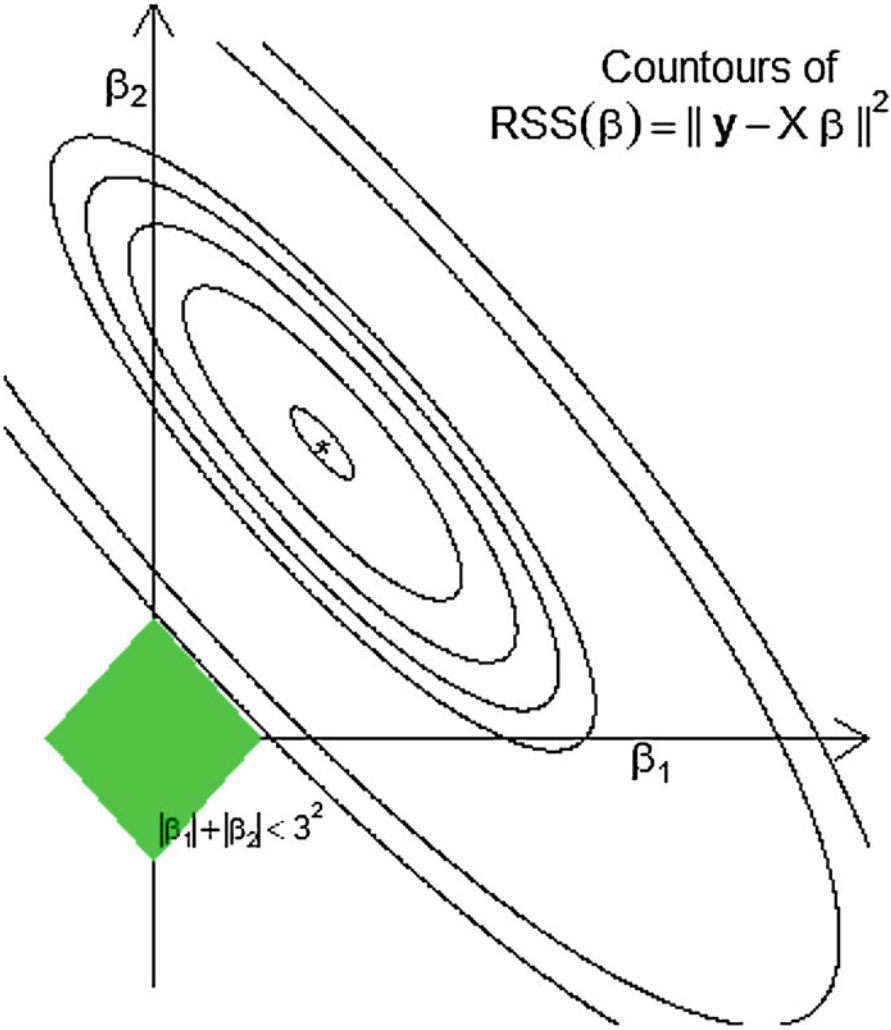

Fig. 3.6 Graphic representation of the Lasso solution of the OLS with restriction $\sum_{j=1}^{p} |\beta_j| < 3^2$. The green region contains the Lasso solution

region is the restriction with $t = 3^2$, which contains the Lasso solution. Indeed, the Lasso solution is the first point of the contours that touches the square, and this will sometimes be in a corner that makes some coefficients zero. Because there are no corners in Ridge regression, this will rarely happen (Tibshirani 1996).

The Lasso estimator can also be derived from a Bayesian perspective. Supposing that the vector of residuals is distributed as $N_n(\mathbf{0}, \sigma^2 \boldsymbol{I}_n)$, like in the Ridge regression case, and assuming that the priors of $\boldsymbol{\beta}_{0s}$ are independent and identically distributed according to Laplace distribution with mean 0 and variance σ_β^2, and adopting a "flat" prior for μ, with known σ^2 and σ_β^2, the posterior distribution of $\boldsymbol{\beta}$ is

$$
\begin{aligned}
& f(\boldsymbol{\beta}_s | \boldsymbol{y}, \boldsymbol{X}_{1s}) \\
& \propto \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}_s\boldsymbol{\beta}_s)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}_s\boldsymbol{\beta}_s)\right] \prod_{i=1}^{n} \exp\left(-\frac{\sqrt{2}}{\sigma_\beta^2}|\beta_{js}|\right) \\
& \propto \exp\left\{-\frac{1}{2\sigma^2}\left[\sum_{i=1}^{n}\left(y_i - \mu - \sum_{j=1}^{p} x_{ijs}\beta_{js}\right) + \lambda \sum_{j=1}^{p}|\beta_{js}|\right]\right\},
\end{aligned}
$$

where $\lambda = \sqrt{8}\sigma^2/\sigma_\beta^2$. Then, the model of the posterior distribution of $\boldsymbol{\beta}_s$ corresponds to the Lasso estimator described before, $\widetilde{\boldsymbol{\beta}}^L(\lambda)$.

The performance of Lasso regression in terms of prediction error is sometimes comparable to Ridge regression (Hastie et al. 2009). However, as we pointed out before, and based on the nature of the restriction term, for any given value of t, only a subset of the coefficients β_{js} is nonzero, so this gives a sparse solution (Efron et al. 2004).

Example 4 To illustrate Lasso regression, here we considered the data used in Example 2, but instead of using a five-fold cross-validation (5FCV) to explore the behavior of this, we built 100 random splittings of the complete data set: 80% for training and 20% for testing. Figure 3.7 presents a representation of the MSE of the Lasso regression (y-axis) and the MSE corresponding to Ridge regression (x-axis). In 81 out of 100 random splittings, the Ridge regression approach gives a better performance, and in this case, on average, the Lasso regression shows an MSE that is 92.13% greater than the Ridge solution. In the other cases, the Ridge was worse, on average, by 30.91%.

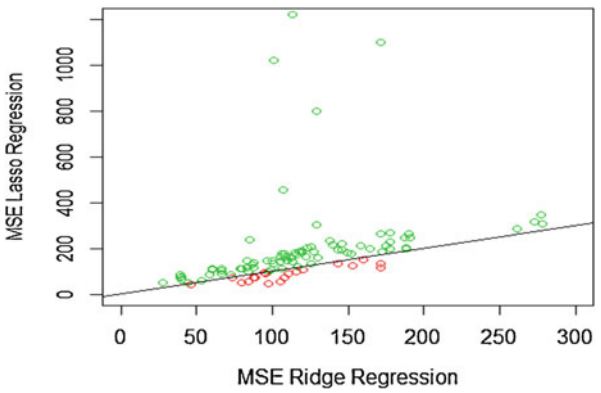

Fig. 3.7 MSE of Ridge regression versus MSE of Lasso regression in 100 random splittings of data: 20% for testing and 80% for training

On average across all the splittings, the performance of the Ridge regression (average = 118.9726 and standard deviation = 50.7193 of MSE) was superior to the Lasso (200.6021 and standard deviation = 222.5494 of MSE) by 68.61%, but this was better than the OLS solution (average = 1609.4635 and standard deviation = 1105.4434 of MSE) by 802.32%, while the Ridge was 1352.80% better than the OLS estimate.

```
#########################R code for Example 4#########################
rm(list=ls())
library(BMTME)
data("WheatMadaToy")
dat_F = phenoMada
dim(dat_F)
dat_F$GID = as.character(dat_F$GID)
G = genoMada
eig_G = eigen(G)
G_0.5 = eig_G$vectors%*%diag(sqrt(eig_G$values))%*%t(eig_G$vectors)
X = G_0.5
y = dat_F$PH
n = length(y)
library(glmnet)
#5FCV
set.seed(3)
K = 5
Tab = data.frame()
set.seed(1)
for(k in 1:100)
{
 Pos_tr = sample(1:n,n*0.8)
 y_tr = y[Pos_tr]; X_tr = X[Pos_tr,]; n_tr = dim(X_tr)[1]
 y_tst = y[-Pos_tr]; X_tst = X[-Pos_tr,]
 #Partition for internal training the model
 Grpv_k = findInterval(cut(sample(1:n_tr,n_tr),breaks=5),1:n_tr)
 #RR
 A_RR = cv.glmnet(X_tr,y_tr,alpha=0,foldid=Grpv_k,type.
measure='mse')
 yp_RR = predict(A_RR,newx=X_tst,s='lambda.min')
 #LR
 A_LR = cv.glmnet(X_tr,y_tr,alpha=1,foldid=Grpv_k,type.
measure='mse')
 yp_LR = predict(A_LR,newx=X_tst,s='lambda.min')
 #OLS
 A_OLS = glmnet(X_tr,y_tr,alpha=1,lambda=0)
 yp_OLS = predict(A_OLS,newx=X_tst)

 Tab = rbind(Tab,data.frame(PT=k,MSEP_RR = mean((y_tst-yp_RR)^2),
             MSEP_LR = mean((y_tst-yp_LR)^2),
             MSEP_OLS = mean((y_tst-yp_OLS)^2)))
 cat('k = ', k,'\n')
}
Tab
```

Now the key components of the just given R code are

1. Hundred random partitions were implemented where each partition is obtained with Pos_tr = sample(1:n,n*0.8), which means that 80% of the data is used for training and 20% for testing, and for each training set, an inner K=5 fold CV is performed to tune the λ hyperparameter.
2. The Grpv_k contains the information of the K=5 fold inner CV implemented to tune the hyperparameter λ.
3. Now we use the cv.glmnet function that is useful for implementing supervised learning methods with cross-validation. This function belongs to the R package glmnet and the input we give to this function is the training set (X_tr, y_tr), alpha=0, that tells glmnet to implement a Ridge regression method, while alpha=1 orders glmnet to implement a Lasso regression. In foldid=Grpv_k we are given training and testing sets to tune the hyperparameter λ, and in type. measure='mse', we are specifying the metric with which we will evaluate the prediction performance of the inner testing sets to be able to choose the best hyperparameter.
4. The function glmnet with lambda=0 implements the OLS estimator.

It is important to point out that Lasso regression performs particularly well when there is a subset of true coefficients that are small or even zero. It doesn't do as well when all of the true coefficients are moderately large; however, in this case, it can still outperform linear regression over a pretty narrow range of (small) λ values.

3.7 Logistic Regression

The logistic regression is a useful and traditional tool used to explain or predict a binary response based on information of explanatory variables. It models the conditional distribution of the response variable as a Bernoulli distribution with the probability of success given by

$$P(Y_i = 1|\boldsymbol{x}_i) = p(\boldsymbol{x}_i;\boldsymbol{\beta}) = \frac{\exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}{1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}.$$

To estimate parameters under logistic regression, suppose that we have a set of data $\left(\boldsymbol{x}_i^{\mathrm{T}}, y_i\right)$, $i = 1, \ldots, n$ (training data), where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}}$ is a vector of features measurement and y_i is the response measurement corresponding to the ith drawn individual. To obtain the MLE of $\boldsymbol{\beta}$, first we need to build the likelihood function of the parameters of $\boldsymbol{\beta}$. This is given by

$$\begin{aligned} L(\boldsymbol{\beta};\boldsymbol{y}) &= \prod_i^n p(\boldsymbol{x}_i;\boldsymbol{\beta})^{y_i}[1 - p(\boldsymbol{x}_i;\boldsymbol{\beta})]^{1-y_i} = \prod_i^n \left(\frac{p(\boldsymbol{x}_i;\boldsymbol{\beta})}{1 - p(\boldsymbol{x}_i;\boldsymbol{\beta})}\right)^{y_i}[1 - p(\boldsymbol{x}_i;\boldsymbol{\beta})]^1 \\ &= \exp\left(\sum_{i=1}^n y_i\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)\right)\prod_{i=1}^n \frac{1}{1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}. \end{aligned}$$

and from here the log-likelihood is

$$\ell(\boldsymbol{\beta};\boldsymbol{y}) = \log\left[L(\boldsymbol{\beta};\boldsymbol{y})\right] = \sum_{i=1}^{n} y_i\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right) - \sum_{i=1}^{n} \log\left[1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)\right].$$

Then, because the gradient of the likelihood is given by

$$\begin{aligned}\frac{\partial \ell(\boldsymbol{\beta};\boldsymbol{y})}{\partial \boldsymbol{\beta}} &= \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} y_i x_{i1} \\ \vdots \\ \sum_{i=1}^{n} y_i x_{ip} \end{bmatrix} - \begin{bmatrix} \sum_{i=1}^{n} \frac{\exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}{1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)} \\ \sum_{i=1}^{n} \frac{\exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}{1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)} x_{i1} \\ \vdots \\ \sum_{i=1}^{n} \frac{\exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}{1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)} x_{ip} \end{bmatrix} \\ &= \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta}) \\ &= \boldsymbol{X}^{\mathrm{T}}[\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta})],\end{aligned}$$

where $\boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta}) = [p(\boldsymbol{x}_1;\boldsymbol{\beta}), \ldots, p(\boldsymbol{x}_n;\boldsymbol{\beta})]^{\mathrm{T}}$, the MLE of $\boldsymbol{\beta}$, $\widehat{\boldsymbol{\beta}}$, can be iteratively approximated by using the gradient descent method:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \alpha \boldsymbol{X}^{\mathrm{T}}[\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta}_t)].$$

For inferential purposes, we have that the asymptotic distribution of $\widehat{\boldsymbol{\beta}}$ is a multivariate normal distribution with vector mean $\boldsymbol{\beta}$ and variance–covariance matrix, the inverse of the negative of the expected value of the Hessian of the log-likelihood, $E\left\{\left[-\frac{\partial \ell(\boldsymbol{\beta};\boldsymbol{y})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^{\mathrm{T}}}\right]^{-1}\right\} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}\right)^{-1}$ (McCullagh and Nelder 1989). This is because

$$\begin{aligned}\frac{\partial^2 \ell(\boldsymbol{\beta};\boldsymbol{y})}{\partial \beta_j \partial \beta_k} = \frac{\partial \ell(\boldsymbol{\beta};\boldsymbol{y})}{\partial \beta_j} &= -\sum_{i=1}^{n} \frac{\exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)}{\left[1 + \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)\right]^2} x_{ij} x_{ik} \\ &= -\sum_{i=1}^{n} p(\boldsymbol{x}_i;\boldsymbol{\beta})[1 - p(\boldsymbol{x}_i;\boldsymbol{\beta})] x_{ij} x_{ik}\end{aligned}$$

The Hessian of the log-likelihood is given by

$$\frac{\partial \ell(\boldsymbol{\beta};\boldsymbol{y})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^{\mathrm{T}}} = \boldsymbol{H} = -\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X},$$

where $\boldsymbol{W} = \mathrm{Diag}\{p(\boldsymbol{x}_1;\boldsymbol{\beta})[1 - p(\boldsymbol{x}_1;\boldsymbol{\beta})], \ldots, p(\boldsymbol{x}_n;\boldsymbol{\beta})[1 - p(\boldsymbol{x}_n;\boldsymbol{\beta})]\}$.

Once the parameters have been estimated, the prediction response is obtained from the estimated probabilities: $\widehat{y}_o = 1$ if $p\left(\boldsymbol{x}_o;\widehat{\boldsymbol{\beta}}\right) > 0.5$ and $\widehat{y}_o = 0$ if

$p\left(\boldsymbol{x}_o;\widehat{\boldsymbol{\beta}}\right) \leq 0.5$. Of course, a different threshold to 0.5 can be used and this could be considered as a hyperparameter that needs to be tuned in a similar fashion as the penalty parameter in the Ridge procedure.

It is important to point out that the minimization process of the log-likelihood can be performed using a more efficient iterative technique called the Newton–Raphson technique, which is an iterative optimization technique that uses a local quadratic approximation to the log-likelihood function. The following is the Newton–Raphson iterative equation used to search for the beta coefficients:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \boldsymbol{H}^{-\mathbf{1}}\boldsymbol{X}^{\mathrm{T}}[\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta}_t)],$$

where $\boldsymbol{H} = -\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}$ is the Hessian matrix whose elements comprise the second derivative of the log-likelihood with regard to the beta coefficients. Therefore, the inverse of the Hessian is $\boldsymbol{H}^{-\mathbf{1}} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X})^{-\mathbf{1}}$. So, the previous equation can be expressed as

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}\right)^{-\mathbf{1}}\boldsymbol{X}^{\mathrm{T}}[\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta}_t)].$$

It is important to recall that the Hessian is no longer constant since it depends on $\boldsymbol{\beta}$ through the weighting matrix $\boldsymbol{W}$. Also, it is clear that the logistic regression does not have a closed solution due to the nonlinearity of the logistic sigmoid function. It is important to point out that if instead of maximizing the likelihood we minimize the negative of the log-likelihood, the Newton–Raphson equation for updating the beta coefficients is equal to

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}\right)^{-\mathbf{1}}\boldsymbol{X}^{\mathrm{T}}[\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{X};\boldsymbol{\beta}_t)].$$

This Newton–Raphson algorithm for logistic regression is known as the iterative reweighted least squares since the diagonal weighting matrix $\boldsymbol{W}$ is interpreted as variances. Another alternative method for estimating the beta coefficients in logistic regression is the Fisher scoring method that is very similar to the Newton–Raphson method just described, but with the difference that instead of using the Hessian ($\boldsymbol{H}$), it uses the expected value of the Hessian matrix, $E(\boldsymbol{H})$.

3.7.1 Logistic Ridge Regression

Like the MLR, when there is strong collinearity, the variance of the MLE is severely affected and the true effects of the explanatory variables could be falsely identified (Lee and Silvapulle 1988). In a similar fashion as for the MLR, this could be judged directly from the asymptotic covariance matrix of $\widehat{\boldsymbol{\beta}}$. Moreover, in a common prediction context, when the number of features is larger than the number of

observations ($p \gg n$), the matrix design is not of full column rank and can cause overfitting, affecting the expected classification error (generalization error) when using the "MLE." One way to avoid overfitting is by replacing the MLE with a regularized MLE as the Ridge MLE estimator of MLR. This is defined as

$$\widetilde{\boldsymbol{\beta}}_s^R(\lambda) = \underset{\boldsymbol{\beta}_s}{\operatorname{argmax}} \left[\ell(\boldsymbol{\beta}_s; \boldsymbol{y}\ ) - \lambda \sum_{j=1}^{p} \beta_{js}^2 \right],$$

where λ is a hyperparameter that has a similar interpretation as in the MLR.

In the literature, there are some algorithms that approximate the Ridge estimation. For example, Genkin et al. (2007) used a cyclic coordinate descent optimization algorithm to approximate this. The one-dimensional optimization problem involved is solved by a modified Newton–Raphson method. Another method was proposed by Friedman et al. (2008) in a more general context. Given the current values of $\widetilde{\boldsymbol{\beta}}_s(\lambda)$, the next update of coordinate β_k is given by

$$\beta_{ks} = \frac{\sum_{i=1}^{n} w_i y_{ij}^* x_{ij}}{\sum_{i=1}^{n} w_i x_{ij}^2 + \lambda}$$

with $y_{ij}^* = y_i^* - \widetilde{\mu}(\lambda) - \sum_{\substack{j=1 \\ j \neq k}}^{p} x_{ijs} \widetilde{\beta}_{js}(\lambda)$ for $k = 1, \ldots, p$, and of μ is given by

$$\mu = \frac{\sum_{i=1}^{n} w_i e_i^*}{\sum_{i=1}^{n} w_i}$$

with $e_i^* = y_i^* - \sum_{j=1}^{p} x_{ijs} \widetilde{\beta}_{js}(\lambda)$, where $y_i^* = \widetilde{\beta}_0(\lambda) + \boldsymbol{x}_i^{\mathrm{T}} \widetilde{\boldsymbol{\beta}}_{0s}(\lambda) + \frac{y_i - p\left(\boldsymbol{x}_i; \widetilde{\boldsymbol{\beta}}_s(\lambda)\right)}{w_i}$ and $w_i = p\left(\boldsymbol{x}_i; \widetilde{\boldsymbol{\beta}}_s(\lambda)\right)\left[1 - p\left(\boldsymbol{x}_i; \widetilde{\boldsymbol{\beta}}_s(\lambda)\right)\right]$, $i = 1, \ldots, n$, are pseudo responses and weights that change across the updates. This can be obtained by maximizing, with respect to β_{ks}, the next quadratic approximation of the penalized likelihood at the current values of $\boldsymbol{\beta}_s$ ($\widetilde{\boldsymbol{\beta}}_s(\lambda)$)

$$\ell(\boldsymbol{\beta}_s; \boldsymbol{y}) - \lambda \sum_{j=1}^{p} \left|\beta_{js}\right| \approx \ell^*(\boldsymbol{\beta}_s; \boldsymbol{y}) - \frac{\lambda}{2} \sum_{j=1}^{p} \beta_{js}^2 + c,$$

where $\ell^*(\boldsymbol{\beta}_s; \boldsymbol{y}) = -\frac{1}{2} \sum_{i=1}^{n} w_i \left(y_i^* - \mu - \sum_{j=1}^{p} x_{ijs} \beta_{js} \right)^2$ is the quadratic approximation of $\ell(\boldsymbol{\beta}_s; \boldsymbol{y})$ at current values of $\boldsymbol{\beta}_s$, $\widetilde{\boldsymbol{\beta}}_s(\lambda)$, and c is a constant that does not depend on $\boldsymbol{\beta}_s$. More details of this implementation (glmnet R package) can be found in Friedman et al. (2008). Note that under this approximation, the beta coefficients can be updated by

$$\widehat{\boldsymbol{\beta}}_s^R(\lambda) = \left(\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{W}\boldsymbol{X}_s + \boldsymbol{\lambda}\boldsymbol{D}\right)^{-1}\boldsymbol{X}_s^{\mathrm{T}}\boldsymbol{W}\boldsymbol{y},$$

where $\boldsymbol{W} = \mathrm{Diag}(w_1, \ldots, w_n)$ and $\boldsymbol{D}$ is the diagonal matrix as defined before.

3.7.2 *Lasso Logistic Regression*

The Lasso penalization can be applied to other models (Tibshirani 1996). In particular, for logistic regression, the Lasso estimator of $\boldsymbol{\beta}_s$ is defined as

$$\widetilde{\boldsymbol{\beta}}_s^L(\lambda) = \underset{\boldsymbol{\beta}_s}{\mathrm{argmax}}\ell_L(\boldsymbol{\beta}_s;\boldsymbol{y}),$$

where $\ell_L(\boldsymbol{\beta}_s;\boldsymbol{y}) = \ell(\boldsymbol{\beta}_s;\boldsymbol{y}) - \lambda\sum_{j=1}^p |\beta_{js}|$ and is often known as the regularized Lasso likelihood. Numerical methods are also required to obtain this Lasso estimate. There are several possibilities (Genkin et al. 2007), such as non-quadratic programming and iteratively reweighted least squares (Tibshirani 1996), but here we will briefly describe the one proposed by Friedman et al. (2008) and implemented in the glmnent R package. This method consists of applying the coordinate descent procedure to a penalized reweighted least square, which is formed by making a Taylor approximation to the likelihood around the current values of the coefficients. That is, this procedure consists of successively updating the parameters by

$$\widetilde{\boldsymbol{\beta}}_s = \underset{\boldsymbol{\beta}_s}{\mathrm{argmin}}\left(-\ell^*(\boldsymbol{\beta}_s;\boldsymbol{y}) + \frac{\lambda}{2}\sum_{j=1}^p |\beta_{js}|\right),$$

where $\ell^*(\boldsymbol{\beta}_s;\boldsymbol{y}) = -\frac{1}{2}\sum_{i=1}^n w_i\left(y_i^* - \mu - \sum_{j=1}^p x_{ijs}\beta_{js}\right)^2 + c$, $y_i^* = \widetilde{\beta}_0(\lambda) + \boldsymbol{x}_i^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}_{0s}(\lambda) + \frac{y_i - p\left(\boldsymbol{x}_i;\widetilde{\boldsymbol{\beta}}_s(\lambda)\right)}{w_i}$, and $w_i = p\left(\boldsymbol{x}_i;\widetilde{\boldsymbol{\beta}}_s(\lambda)\right)\left[1 - p\left(\boldsymbol{x}_i;\widetilde{\boldsymbol{\beta}}_s(\lambda)\right)\right]$, $i = 1, \ldots, n$, as defined in the Ridge regression case. More details of this implementation can be consulted in Friedman et al. (2008).

Example 5 In this example, we used data corresponding to 40 lines planted with four repetitions. For illustrative purposes, we will use as response a binary variable based on Plant Height. The matrix of features used here was obtained from the genomic relationship ($\boldsymbol{G}$), $\boldsymbol{X} = \boldsymbol{Z}_L\boldsymbol{G}^{1/2}$, where $\boldsymbol{G}^{1/2}$ is the square root matrix of $\boldsymbol{G}$.

The performance of logistic regression, logistic Ridge regression, and Lasso logistic regression for this data set was evaluated across 100 random splittings of the complete data set: 20% for testing (evaluation performance) and 80% for training. The performance was measured by the proportion of cases correctly classified (PCCC) in the testing data. These results are summarized in Table 3.2,

Table 3.2 Performance of the standard, Ridge, and Lasso logistic regression models

Method	PCCC	SD
SLR	0.7284	0.0765
Ridge	0.7240	0.0763
Lasso	0.7206	0.0705

PCCC denotes the proportion of cases correctly classified

where for each method the mean (PCCC) and standard deviation (SD) of the PCCC across the 100 splittings are reported. The table indicates that, on average, the standard logistic (SLR) approach shows slightly better performance than the other two approaches, even better than the Lasso solution. Out of the 100 random partitions, 72, 24, and 4, the SLR, the logistic Ridge regression (LRR), and the logistic lasso regression (LLR) resulted in the higher PCCC value, respectively. However, the difference in the performance of the three methods is not significant because of the large deviation obtained across the different partitions.

The computations were done with the help of the glmnet R package using the following R code:

```
##########################R code for Example 5#######################
load(file ='dat-E3.5.RData')
dat_F = dat$dat_F
dat_F = dat_F[order(dat_F$Rep,dat_F$GID),]
head(dat_F)
G = dat$G
dat_F$y = dat_F$Height
ZL = model.matrix(~0+GID,data=dat_F)
colnames(ZL)
Pos = match(colnames(ZL),paste('GID',colnames(G),sep=''))
max(abs(diff(Pos)))
y = dat_F$y
ei = eigen(G)
X = ZL%*%ei$vectors%*%diag(sqrt(ei$values))%*%t(ei$vectors)
n = length(y)
library(glmnet)
#5FCV
set.seed(1)
Tab = data.frame()
set.seed(1)
for(k in 1:100)
{
 Pos_tr = sample(1:n,n*0.8)
 y_tr = y[Pos_tr] ; X_tr = X[Pos_tr,]; n_tr = dim(X_tr)[1]
 y_tst = y[-Pos_tr]; X_tst = X[-Pos_tr,]
 #Partition for internal training the model
 Grpv_k = findInterval(cut(sample(1:n_tr,n_tr),breaks=5),1:n_tr)
 #RR
 A_RR = cv.glmnet(X_tr,y_tr,family='binomial',
          alpha=0,foldid=Grpv_k,type.measure='class')
 yp_RR = as.numeric(predict(A_RR,newx=X_tst,s='lambda.min',
type='class'))
```

```
 #LR
 A_LR = cv.glmnet(X_tr,y_tr,family='binomial',
         alpha=1,foldid=Grpv_k,type.measure='class')
 yp_LR = as.numeric(predict(A_LR,newx=X_tst,s='lambda.min',
type='class'))
 #SLR
 A_SLR = glmnet(X_tr,y_tr,family='binomial',alpha=0,lambda=0)
 yp_SLR = as.numeric(predict(A_SLR,newx=X_tst,type='class'))

 Tab = rbind(Tab,data.frame(PT=k,PCCC_RR = 1-mean(y_tst!=yp_RR),
               PCCC_LR = 1-mean(y_tst!=yp_LR),
               PCCC_SLR = 1-mean(y_tst!=yp_SLR)))
 cat('k = ', k,'\n')
}
Tab
```

Also, in this R code there are four relevant points:

1. Hundred random partitions were implemented, Pos_tr = sample(1:n,n*0.8), with 80% for training and 20% for testing, and for each training set, an inner K=5 fold CV is performed to tune the λ hyperparameter.
2. Also, here Grpv_k contains the information of the K=5 inner folds to tune the hyperparameter λ.
3. The cv.glmnet function with the following input is used: (a) training set (X_tr, y_tr); (b) with alpha=0 to implement a Ridge regression and alpha=1 to implement a Lasso regression; (c) with foldid=Grpv_k containing training and testing sets to tune the hyperparameter λ; (d) with family='binomial' to implement a logistic regression; and (e) with type.measure='class' as a metric for categorical data to measure the prediction performance of the inner testing set to choose the best value of the hyperparameter λ.
4. The function glmnet with family='binomial' and lambda=0 implements the logistic regression but without penalization.

Appendix 1: R Code for Ridge Regression Used in Example 2

The TR_RR.R script file must contain the following:

```
library(epiR)
source("RR.R")
Tr_RR_f<-function(y_tr,X_tr,K=5,KG=100,KR=1)
{
 n_tr = dim(X_tr)[1]
 X_tr_s = scale2_f(X_tr)
 mu   = mean(y_tr)
 y_tr = y_tr-mu
 #Inner CV
 lambv = lamb_f(X_tr,K=KG,li=1e-7,ls=1-1e-7)
```

```
 iPEv_mean = 0
 for(ir in 1:KR)
 {
  iPE_mat = matrix(0,nr=length(lambv),nc=K)
  MSEP_mat=iPE_mat
  Grpv = findInterval(cut(sample(1:n_tr,n_tr),breaks=K),1:n_tr)
  for(i in 1:K)
  {
   Pos_itr = which(Grpv!=i)
   X_itr  = X_tr_s[Pos_itr,]
   y_itr  = y_tr[Pos_itr];
   y_itst  = y_tr[-Pos_itr];
   dat_itr = data.frame(y=y_itr,X=X_itr)
   n_itr =  dim(X_itr)[1]
   betav_itr = A_RR_f_a_V(y_itr,X_itr,lambv)
   yp_mat   = 0 + (X_tr_s[-Pos_itr,])%*%betav_itr
   iPEv    = colMeans((matrix(y_itst,nr=length(y_itst),
                 nc=length(lambv))-yp_mat)**2)
   iPE_mat[,i] =iPEv
  }
  iPEv_mean = (ir-1)/ir*iPEv_mean + rowMeans(iPE_mat)/ir

 }

 #
 plot(log(lambv),iPEv_mean)
 Pos_o = which.min(iPEv_mean)
 lamb_o = lambv[Pos_o]
 A_RR = RR_f(y=y_tr+mu,X=X_tr,lamb=lamb_o,Intercept = TRUE,Std=TRUE)
 betav_s = A_RR$betav_s
 betav_o = A_RR$betav_o
 list(lamb_o = lamb_o, betav_s =  betav_s,
    betav_o = betav_o,
    lambv=lambv,iPEv_mean=iPEv_mean,X_tr=X_tr,y_tr =y_tr,Grpv=Grpv)
}

Pred_RR_f<-function(y_tst,X_tst,TR_RR)
{
 #betava_RR = TR_RR$betava_RR
 y_tr = TR_RR$y_tr; X_tr = TR_RR$X_tr
 X   = rbind(X_tr,X_tst)
 betav_s = TR_RR$betav_s
 betav_o = TR_RR$betav_o
 yp_tst  = c(cbind(1,X_tst)%*%betav_o)
 plot(y_tst,yp_tst); abline(a=0,b=1)
 MSEP_RR =  mean((y_tst-yp_tst)**2)
 list(MSEP=MSEP_RR, betav_s = betav_s,betav_o = betav_o)
}
Pred_ols_f<-function(y_tst,X_tst,y_tr,X_tr)
{
 #OLS
 p =  dim(X_tr)[2]
 A = lm_f(y_tr,X_tr)
```

```
 sdv  = apply(X_tr,2,sd2_f)
 A_inv = diag(c(1,1/sdv))
 A_inv[1,1] = 1
 A_inv[1,2:(p+1)] = -apply(X_tr,2,mean)/sdv
 betav_s = A$betav_s
 betav_o = A_inv%*%betav_s
 yp_tst_ols = c(cbind(1,X_tst)%*%betav_o)
 MSEP_ols = mean((y_tst-yp_tst_ols)**2)
 list(MSEP=MSEP_ols,betav_s = betav_s)
}
```

The script file accessed by source("RR.R") must contain the following R code:

```
sd2_f<-function(x)
{
 (mean((x-mean(x))**2))^0.5
}
scale2_f<-function(X)
{
 scale(X,center=TRUE,scale = apply(X,2,sd2_f))
}

#RR internal standarized: zij = (xij-\bar{x}_j)/ss_j
#ss_j = mean (xij-\bar{x}_j)^2
RR_f<-function(y,X,lamb = 0, Intercept=TRUE,Std=TRUE)
{
 p   = dim(X)[2]; n = dim(X)[1]
 if(Std == TRUE)
 {
  sdv  = apply(X,2,sd2_f)
  A_inv = diag(c(1,1/sdv))
  A_inv[1,1] = 1
  A_inv[1,2:(p+1)] = -apply(X,2,mean)/sdv
  mu = mean(y)
  X_s = scale2_f(X)
  Xa = X_s
  svd_X = svd(Xa); d1  = svd_X$d
  Gama1 = svd_X$v
  U = svd_X$u
  pa   = dim(Gama1)[2]
  betav_s = c(mu,Gama1%*%(((d1/(d1^2+lamb)))*(t(U)%*%(y-mu))))
  betav_o = A_inv%*%betav_s# betav in orginal scale
  list(betav_s = betav_s,betav_o=betav_o)
 }
 else
 {
  Xa = X
  sdv  = apply(X,2,sd2_f)
  p = dim(X)[2]
  svd_X = svd(Xa)
  d1  = svd_X$d
  Gama1 = svd_X$v; U = svd_X$u
  betav = Gama1%*%((d1/(d1^2+lamb))*(t(U)%*%(y)))#-mean(y)
```

```
  betav
  #tX = t(X)
  #betav = ginv(tX%*%X+lamb*diag(p))%*%tX%*%y
  #betav
 }
}

RR_f_V = Vectorize(RR_f,'lamb')

A_RR_f_a <-function(y_itr,X_itr,lamb)
{
 A = RR_f(y=y_itr,X=X_itr,lamb=lamb,Std=FALSE,Intercept = FALSE)
 A
}
A_RR_f_a_V<-Vectorize(A_RR_f_a,'lamb')

lamb_f<-function(X,K=100,li=0.001,ls=0.999)
{
 Xac = scale2_f(X)
 n = dim(Xac)[1]
 R2v = seq(li,ls,length=K)
 lambv = (1-R2v)/R2v*sum(diag(Xac%*%t(Xac)))/n
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}

library(MASS)
lm_f<-function(y,X)
{
 p = dim(X)[2]
 sdv  = apply(X,2,sd2_f)
 A_inv = diag(c(1,1/sdv))
 A_inv[1,1] = 1
 A_inv[1,2:(p+1)] = -apply(X,2,mean)/sdv
 X = scale2_f(X); mu = mean(y);
 svd_X = svd(X); d = svd_X$d
 d1  = svd_X$d; Gama1 = svd_X$v; U = svd_X$u
 betav_s = c(mu,Gama1%*%((1/d1)*(t(U)%*%(y-mu))))
 betav_o = A_inv%*%betav_s# betav in orginal scale
 list(betav_s = betav_s,betav_o=betav_o)
}
```

References

Allaire JJ, Chollet F (2019) keras: R Interface to 'Keras'. R package version, 2(4)
Beysolow T II (2017) Introduction to deep learning using R: a step-by-step guide to learning and implementing deep learning models using R. Apress, San Francisco, CA
Burden RL, Faires JD (2011) Numerical analysis. Cengage Learning, Boston, MA
Casella G, Berger RL (2002) Statistical inference. Duxbury, Thomson Learning, Pacific Grove, CA

Christensen P (2011) Plane answers to complex questions: the theory of linear models. Springer Science+Business Media, New York
Efron B, Hastie T, Johnstone I, Tibshirani R (2004) Least angle regression. Ann Stat 32(2):407–499
Friedman J, Hastie T, Hoeing H, Tibshirani R (2007) Pathwise coordinate optimization. Ann Appl Stat 2(1):302–332
Friedman J, Hastie T, Tibshirani R (2008) Sparse inverse covariance estimation with the graphical lasso. Biostatistics 9(3):432–441
Friedman J, Hastie T, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. J Stat Softw 33(1):1–22
Genkin A, Lewis D, Madigan D (2007) Large-scale Bayesian logistic regression for text categorization. Technometrics 49(3):291–304
Goodfellow I, Bengio Y, Courville A (2016) Deep learning. The MIT Press, Cambridge, MA
Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction. Springer, New York
Hastie T, Tibshirani R, Wainwright M (2015) Statistical learning with sparsity: the lasso and generalizations. Chapman and Hall/CRC, New York
Haykin S (2009) Neural networks and learning machines. Pearson Prentice Hall, New Jersey, p 909
James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning: with applications in R. Springer Science+Business Media, New York
Lee AH, Silvapulle MJ (1988) Ridge estimation in logistic regression. Commun Stat Simul Comput 178(4):1231–1257. https://doi.org/10.1080/03610918808812723
McCullagh P, Nelder JA (1989) Generalized linear models. Chapman and Hall, London, England
Montgomery DC, Peck EA, Vining GG (2012) Introduction to linear regression. Wiley, Hoboken, NJ
Nocedal J, Wright SJ (2006) Numerical optimization. Springer, New York
Rencher AC, Schaalje GB (2008) Linear models in statistics. Wiley, Hoboken, NJ
Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc B 58(1):267–288
Wakefield J (2013) Bayesian and frequentist regression methods. Springer Science+Business Media, New York
Warner B, Misra M (1996) Understanding neural networks as statistical tools. Am Stat 50 (4):284–293

Chapter 4
Overfitting, Model Tuning, and Evaluation of Prediction Performance

4.1 The Problem of Overfitting and Underfitting

The *overfitting* phenomenon occurs when the statistical machine learning model learns the training data set so well that it performs poorly on unseen data sets. In other words, this means that the predicted values match the true observed values in the training data set too well, causing what is known as overfitting. Overfitting happens when a statistical machine learning model learns the systematic and noise (random fluctuations) parts in the training data to the extent that it negatively impacts the performance of the statistical machine learning model on new data. This means that the statistical machine learning model adapts very well to the noise as well as to the signal that is present in the training data. The problem is that these concepts do not apply to independent (new) data and negatively affect the model's ability to generalize. Overfitting is more probable when learning a loss function from a complex statistical machine learning model (with more flexibility). For this reason, many nonparametric statistical machine learning models also include constraints in the loss function to improve the learning process of the statistical machine learning models. For example, artificial neural networks (ANN), mentioned later, are a nonparametric statistical machine learning model that is very flexible and is subject to overfitting training data. This problem can be addressed by dropping out (setting to zero) the weights of a certain percentage of hidden units in order to avoid overfitting.

On the other hand, an *underfitted* phenomenon occurs when few predictors are included in the statistical machine learning model, i.e., it is a very simple model that poorly represents the complete picture of the predominant data pattern. This problem also arises when the training data set is too small or not representative of the population data. An *underfitted* model does a poor job of fitting the training data and for this reason it is not expected to satisfactorily predict new data points. This implies that the predictions using unseen data are weak, since individuals are perceived as strangers unfamiliar with the training data set.

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_4

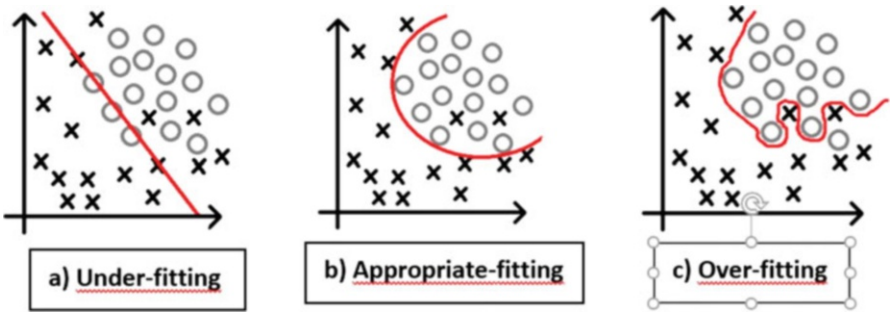

Fig. 4.1 Schematic illustration of three models for classification: (**a**) M1 with underfitting, (**b**) M2 with appropriate fitting, and (**c**) M3 with overfitting

Consider a scattered series of points $(y_1, x_1), \ldots, (y_n, x_n)$, on a plane, to which we want to adjust a statistical machine learning method. This means that we are looking for the best $f(\boldsymbol{x}_i)$ that explains the existing relationship between the response variable (y_i) and the predictors $(x_1, \ldots, x_n)$. We assume that we have three options for $f(\boldsymbol{x}_i)$: M1, the simple model plotted in Fig. 4.1a; M2, an intermediate model shown in Fig. 4.1b; and M3, a complex model shown in Fig. 4.1c.

Under the classification framework, the first panel in Fig. 4.1 (left side, panel a) shows an unsatisfactory fit (underfitted) since the line does not cover most of the points (has high bias) in the plot. As such, we expect that the prediction of unseen data of this model, M1, will perform badly. In contrast, panel c of Fig. 4.1 shows an almost perfect fit, since the predicted line covers all the data points. While at first glance, you may think that model M3 will perform well when predicting unseen data, this is actually untrue since the predicted line covers all points that are noise and those that are signal (overfit); for this reason, this type of model also performs poorly in the prediction of future data due to its complexity and high variance. Therefore, the best option for predicting unseen data is model M2 (Fig. 4.1, panel b) since it represents the predominant (smooth) pattern enough to represent the apparent data pattern while maintaining a balance between bias and variance. For this reason, a well-fitted model is one that faithfully represents the sought-after predominant pattern in the data, while ignoring the idiosyncrasies in the training data. As such, a well-fitted model in the testing set should be in the neighborhood of the model's accuracy based on the training data set, that is, the model's accuracy in the testing set should be approximately equal to that of the model's accuracy in the training set. In contrast, an overfitted model in the testing data set will be far from the neighborhood of the model's accuracy based on the training data set, and usually its prediction performance is very high (good) in the training set and consequently low (bad) in the testing set (Ratner 2017).

The paradox of overfitting is defined as complex models that contain more information about the training data, but less information about the testing data (future data we want to predict). In statistical machine learning, overfitting is a major issue and leads to some serious problems in research: (a) some relationships

that seem statistically significant are only noise, (b) the complexity of the statistical machine learning model is very large for the amount of data provided, and (c) the model in general is not replicable and predicts poorly.

Since the main goal of developing and implementing statistical machine learning methods is to predict unseen data not used for training the statistical machine learning algorithm, researchers are mainly interested in minimizing the testing error (generalization error applicable to future samples) instead of minimizing the training error that is applicable to the observed data used for training the statistical machine learning algorithm.

According to Shalev-Shwartz and Ben-David (2014), if the learning fails, these are some approaches to follow:

1. Increase the sample size of the training set.
2. Modify the hypothesis by (a) enlarging it, (b) reducing it, (c) completely changing it, and (d) changing the parameters being used. We understand a hypothesis as the models and their parameters under evaluation. This point is very important to reach a reasonable model for your data.
3. Change the feature representation of the data.
4. Change the statistical machine learning algorithm used.

4.2 The Trade-Off Between Prediction Accuracy and Model Interpretability

Accuracy is the ability of a statistical machine learning model to make correct predictions and those models with more complexity (called flexible models) are better in terms of accuracy, while the simple, less complex models (called inflexible models) are less accurate but more interpretable. Interpretability indicates to what degree the model allows for human understanding of natural phenomena. For these reasons, when the goal of the study is prediction, flexible models should be used; however, when the goal of the study is inference, inflexible models are more appropriate because they more easily interpret the relationship between the response variables and the predictor variables. As the complexity of the statistical machine learning model increases, the bias is reduced and the variance increases. For this reason, when more parameters are included in the statistical machine learning model, the complexity of the model increases and the variance becomes the main concern while the bias steadily falls. For example, James et al. (2013) state that the linear regression model is a relatively inflexible method because it only generates linear functions, while the support vector machine method is one of the most flexible statistical machine learning methods.

Before providing an analytical interpretation of the trade-off between the bias and variance, we must understand the meaning of both concepts. Bias is the difference between the expected prediction of our statistical machine learning model and the true observed values. For example, assume that you poll a specific city where half of

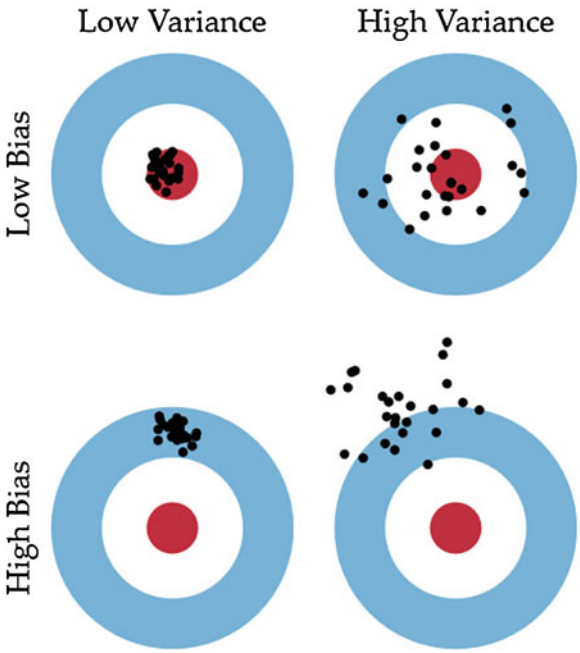

Fig. 4.2 Graphical representations of different levels of bias and variance

the population is high-income and the other half is low-income. If you collected a sample of high-income people, you would conclude that the entire city has high income. This means that your conclusion is heavily biased since you only sampled people with high income. On the other hand, error variance refers to the amount that the estimate of the objective function will change using a different training data set. In other words, the error variance accounts for the deviation of predictions from one repetition to another using the same training set. Ideally, when a statistical machine learning model with low error variance predicts a value, the predicted value should remain almost the same, even when changing from one training data set to another; however, if the model has high variance, then the predicted values of the statistical machine learning method are affected by the values of the data set. We provide a graphical visualization of bias and variance with a bull's eye diagram (see Fig. 4.2). We assume that the center of the target is a statistical machine learning model that perfectly predicts the correct answers. As we move away from the bull's eye, our predictions get worse. Let us assume that we can repeat our entire statistical machine learning model building process to get a number of separate hits on the target. Each hit represents an individual realization of our statistical machine learning model, given the chance variability in the training data we gathered. Sometimes we accurately predict the observations of interest since we captured a representative sample in our training data, while other times we obtain unreliable predictions since our training data may be full of outliers or nonrepresentative values. The four combinations of cases resulting from both high and low bias and variance are shown in Fig. 4.2.

Burger (2018) concludes that the best scenario is the one with *low bias* and *low variance*, since samples are acceptable representatives of the population (Fig. 4.2), while in the case of *high bias* and *low variance*, the samples are fairly consistent, but not particularly representative of the population (Fig. 4.2). However, when there is *low bias* and *high variance*, the samples vary widely in their consistency, and only some may be representative of the population (Fig. 4.2). Finally, with *high bias* and *high variance*, the samples are somewhat consistent, but unlikely to be representative of the population (Fig. 4.2).

If we denote the variable we are trying to predict as y and our covariates as $\boldsymbol{x}_i$, we may assume that there is a relationship between y and $\boldsymbol{x}_i$, as that given in Eq. (1.1) from Chap. 1, where the error term is normally distributed with a mean of zero and variance σ^2. The expected prediction error for a new observation with value x, using a quadratic loss function, is given by Hastie et al. (2008, page 223):

$$\begin{aligned} E\left(y-\widehat{f}\{\boldsymbol{x}_i\}\right)^2 &= E(y-f\{\boldsymbol{x}_i\})^2 + \left(E\left(\widehat{f}\{\boldsymbol{x}_i\}\right)-f\{\boldsymbol{x}_i\}\right)^2 + E\left\{\widehat{f}(\boldsymbol{x}_i)-E\left(\widehat{f}\{\boldsymbol{x}_i\}\right)\right\}^2 \\ &= \mathrm{Var}(y) + \left[\mathrm{Bias}\left(\widehat{f}\{\boldsymbol{x}_i\}\right)\right]^2 + \mathrm{Var}\left(\widehat{f}(\boldsymbol{x}_i)\right) \\ &= \mathrm{Var}(\epsilon) + \left[\mathrm{Bias}\left(\widehat{f}\{\boldsymbol{x}_i\}\right)\right]^2 + \mathrm{Var}\left(\widehat{f}(\boldsymbol{x}_i)\right), \end{aligned}$$

where Bias is the result of misspecifying the statistical model f. Estimation variance (the third term) is the result of using a sample to estimate f. The first term is the error (irreducible error) that results even if the model is correctly specified and accurately estimated. This irreducible error is the noise term in the true relationship that cannot fundamentally be reduced by any model. Given the true model and infinite data to train (calibrate) it, we should be able to reduce both the bias and variance terms to 0. However, in a world with imperfect models and finite data, there is a trade-off between minimizing the bias and minimizing the variance. The above decomposition reveals a source of the difference between explanatory and predictive modeling: In explanatory modeling, the focus is on minimizing bias to obtain the most accurate representation of the underlying theory. In contrast, predictive modeling seeks to minimize the combination of bias and estimation variance, occasionally sacrificing theoretical accuracy for improved empirical precision (Shmueli 2010).

These four aspects impact every step of the modeling process, such that the resulting f is markedly different in the explanatory and predictive contexts.

Let us assume that f is a reasonable operationalization of the true function (F) relating constructs X and Y. Choosing a function f^* that is intentionally biased in place of f is very undesirable from a theoretical–explanatory standpoint. However, the election of f^* is desirable to f under the prediction approach. We show this using the statistical model $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$, which is assumed to be correctly specified with respect to F. Using data, we obtain the estimated model $\widehat{f}$, which has the following properties:

$$\text{Bias} = 0$$

$$\text{Var}\left(\widehat{f}(\boldsymbol{x}_i)\right) = \text{Var}\left(\widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_2 + \widehat{\beta}_3 x_3\right) = \sigma^2 \boldsymbol{x}^{\text{T}} \left(\boldsymbol{X}\boldsymbol{X}^{\text{T}}\right)^{-1} \boldsymbol{x},$$

where $\boldsymbol{x}$ is the vector $\boldsymbol{x} = [x_1, x_2, x_3]^{\text{T}}$ and $\boldsymbol{X}$ is the design matrix based on all predictors. Combining the squared bias with the variance gives, as expected, the prediction error (EPE).

$$E\left(y - \widehat{f}\{\boldsymbol{x}_i\}\right)^2 = \sigma^2 + 0 + \sigma^2 \boldsymbol{x}^{\text{T}} \left(\boldsymbol{X}\boldsymbol{X}^{\text{T}}\right)^{-1} \boldsymbol{x} = \sigma^2 \left[1 + \boldsymbol{x}^{\text{T}} \left(\boldsymbol{X}\boldsymbol{X}^{\text{T}}\right)^{-1} \boldsymbol{x}\right].$$

In comparison, consider the estimated underspecified form $\widehat{f}^*(\boldsymbol{x}_i) = x_1 \widehat{\gamma}$. The bias and variance here are

$$\begin{aligned}\text{Bias} = E\left(\widehat{f}\{\boldsymbol{x}_i\}\right) - f\{\boldsymbol{x}_i\} &= x_1 \left(x_1 x_1^{\text{T}}\right)^{-1} x_1^{\text{T}} (\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) \\ &\quad - (\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)\end{aligned}$$

$$\text{Var}\left(\widehat{f}(\boldsymbol{x}_i)\right) = \sigma^2 x_1 \left(x_1 x_1^{\text{T}}\right)^{-1} x_1^{\text{T}}$$

Combining the squared bias with the variance EPE is equal to

$$\begin{aligned}E\left(y - \widehat{f}\{\boldsymbol{x}_i\}\right)^2 &= \left[x_1 \left(x_1 x_1^{\text{T}}\right)^{-1} x_1^{\text{T}} (x_2\beta_2 + x_3\beta_3) - (x_2\beta_2 + x_3\beta_3)\right]^2 \\ &\quad + \sigma^2 \left[1 + x_1 \left(x_1 x_1^{\text{T}}\right)^{-1} x_1^{\text{T}}\right].\end{aligned}$$

Although the bias of the underspecified model $f^*(\boldsymbol{x}_i)$ is larger than that of $f\{\boldsymbol{x}_i\}$, its variance can be smaller, and in some cases, so small that the overall EPE will be lower for the underspecified model. Wu et al. (2007) showed the general result for an underspecified linear regression model with multiple predictors. In particular, they showed that the underspecified model that leaves out q predictors has a lower EPE when the following inequality holds:

$$q\sigma^2 > \beta_2^{\text{T}} X_2^{\text{T}} (I - H_1) X_2 \beta_2$$

This means that the underspecified model produces more accurate predictions, in terms of lower EPE, in the following situations: (a) when the data are very noisy (large σ^2); (b) when the true absolute values of the excluded parameters (in our example, β_2 and β_3) are small; (c) when the predictors are highly correlated; and (d) when the sample size is small or the range of left-out variables is small (Shmueli 2010).

Hagerty and Srinivasan (1991) nicely summarize this situation: "We note that the practice in applied research of concluding that a model with a higher predictive validity is "truer," is not a valid inference. This paper shows that a parsimonious but less true model can have a higher predictive validity than a truer but less parsimonious model."

4.3 Cross-validation

Cross-validation (CV) is a strategy for model selection or algorithm selection. CV consists of splitting the data (at least once) for estimating the error of each algorithm. Part of the data (the training set) is used for training each algorithm, and the remaining part (the testing set) is used for estimating the error of the algorithm. Then, CV selects the algorithm with the smallest estimated error. For this reason, CV is used to evaluate the prediction performance of a statistical machine learning model in out-of-sample data. This technique ensures that the data used for training the statistical machine learning model are independent of the testing data set in which the prediction performance is evaluated. It consists of repeating and recording the arithmetic average obtained from the evaluation measures on different partitions. Under k-fold CV, which is explained in greater detail later, this process is repeated a total of k times, with each of the k groups getting the chance to play the role of the test data, and the remaining $k - 1$ groups used as training data. In this way, we obtain k different estimates of the prediction error. As prediction performance is reported the average of these estimates of prediction error. CV is used in data analysis to validate the implemented models where the main objective is prediction and to estimate the prediction performance of a statistical learning model that will be carried out in practice. In other words, CV evaluates how well the statistical machine learning model generalized new data not used for training the model. The results of the CV largely depend on how the division between the training and testing sets is carried out. For this reason, in the following sections, we provide the more popular types of CV used in the implementation of statistical learning models.

4.3.1 The Single Hold-Out Set Approach

The single hold-out set or validation set approach consists of randomly dividing the available data set into a training set and a validation or hold-out set (Fig. 4.3). The statistical machine learning model is trained with the training set while the hold-out

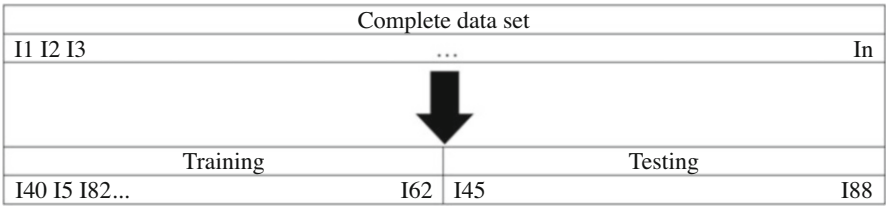

Fig. 4.3 Schematic representation of the hold-out set approach. A set of observations are randomly split into a training set with individuals I40, I5, I82, among others, and into a testing set with observations I45, I88, among others. The statistical machine learning model is fitted on the training set and its performance is evaluated on the validation set (James et al. 2013)

set (testing set) is used to study how well that statistical machine learning model performs on unseen data. For example, 80% of the data can be used for training the model and the remaining 20% of the data for testing it. One weakness of the hold-out (validation) set approach is that it depends on just one training-testing split and its performance depends on how the data are split into the training and testing sets.

4.3.2 *The* k-*Fold Cross-validation*

In k-fold CV, the data set is randomly divided into k complementary folds (groups) of approximately equal size. One of the subsets is used as testing data and the rest $(k - 1)$ as training data. Then $k - 1$ folds are used for training the statistical machine learning model and the remaining fold for evaluating the out-of-sample prediction performance. For these reasons, the statistical machine learning model is fitted k times using a different partition (fold) as the testing set and the remaining $k - 1$ as the training set. Finally, the arithmetic mean of the k folds is obtained and reported as the prediction performance of the statistical machine learning model (see Fig. 4.4). This method is very accurate because it combines k measures of fitness resulting from the k training and testing data sets into which the original data set was divided, but at the cost of more computational resources. In practice, the choice of the number of folds depends on the measurement of the data set, although 5 or 10 folds are the most common choices.

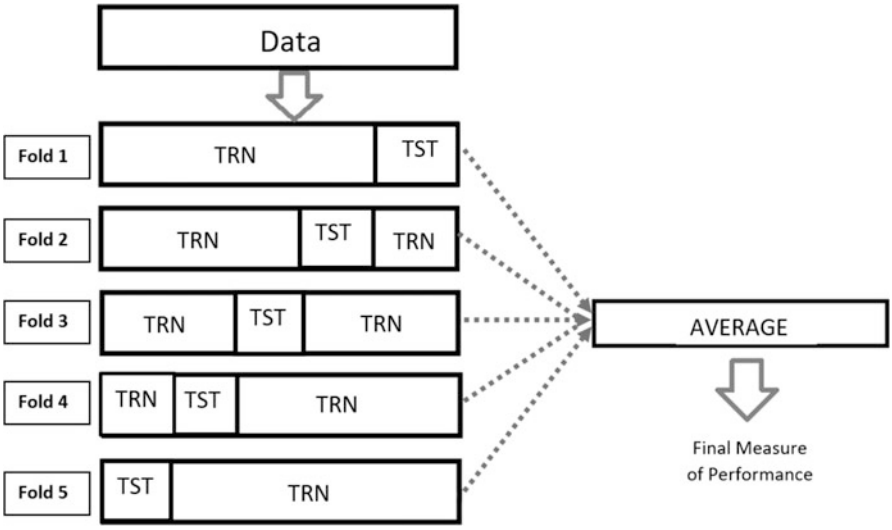

Fig. 4.4 Schematic representation of the k-fold cross-validation with complementary subsets with $k = 5$

It is important to point out that to reduce variability, we recommend implementing the *k*-fold CV multiple times, each time using different complementary subsets to form the folds; the validation results are combined (e.g., averaged) over the rounds (times) to give a better estimate of the statistical machine learning model predictive performance.

4.3.3 The Leave-One-Out Cross-validation

The Leave-One-Out (or LOO) CV is very simple since each training data set is created by including all the individuals except one, while the testing set only includes the excluded individual. Thus, for n individuals in the full data set, we have n different training and testing sets. This CV scheme wastes minimal data, as only one individual is removed from the training set.

Regarding the *k*-fold cross-validation that was just explained, n models are built from n individuals (samples) instead of k models, where $n > k$. Moreover, each model is trained on $n - 1$ samples rather than $(k - 1)n/k$. In both cases, since k is normally not too large and $k < n$, LOO is more computationally expensive than *k*-fold cross-validation. In terms of prediction performance, LOO normally produces high variance for the estimation of the test error. Because $n - 1$ of the n samples is used to build each statistical machine learning model, those constructed from folds are virtually identical to each other and to the model built from the entire training set. However, when the learning curve is steep for the evaluated training size, then five- or ten-fold cross-validation usually overestimates the generalization error.

Learning curves (LC) are considered effective tools to monitor the performance of the employee exposed to a new task. LCs provide a mathematical representation of the learning process that takes place as the task is repeated. In statistical machine learning the LC is a line plot of learning (y-axis) over experience (x-axis). Learning curves are extensively used in statistical machine learning for algorithms that learn (their parameters) incrementally over time, such as deep neural networks. In general, there is considerable empirical evidence suggesting that five- or ten-fold cross-validation should be preferred to LOO.

4.3.4 The Leave-m-Out Cross-validation

The Leave-m-Out (LmO) CV is very similar to LOO as it creates all the possible training/test sets by removing m samples from the complete set. For n samples, this produces $\binom{n}{m}$ train-test pairs. Unlike LOO and *k*-fold, the test sets will overlap for $m > 1$. For example, in a Leave-2-Out CV with a data set with four samples (I1, I2,

I3, and I4), the total number of training-testing sets is equal to $\binom{4}{2} = 6$; this means that the six testing sets are: [I3, I4] [I1, I2], [I2, I4] [I1, I3], [I2, I3] [I1, I4], [I1, I4] [I2, I3],[I1, I3] [I2, I4], and [I1,I2] [I3, I4], while the training sets are the complementary elements of each testing set.

4.3.5 Random Cross-validation

In this type of CV, the number of partitions (independent training-testing data set splits) is defined by the user, and more partitions are better. Each partition is generated by randomly dividing the whole data set into two subsets: the training (TRN) data set and the testing (TST) data set. The percentage of the whole data set assigned to the TRN and TST data sets is also fixed by the user. For example, for each random partition, the user can decide that 80% of the whole data set can be assigned to the TRN data set and the remaining 20% to the TST data set. Random cross-validation is different from *k*-fold cross-validation because the partitions are not mutually exclusive; this means that in the random cross-validation approach, one observation can appear in more than one partition. Consequently, some samples cannot be evaluated, whereas others can be evaluated more than once, meaning that the testing and training subsets can be superimposed (Montesinos-López et al. 2018a, b). To control the randomness for reproducibility, we recommend using a specific seed in the random number generator. We recommend using at least ten random partitions to obtain enough accuracy in the estimate of prediction performance.

4.3.6 The Leave-One-Group-Out Cross-validation

The Leave-One-Group-Out (LOGO) CV is useful when individuals are grouped (in environments or years, or even another criterion), where the number of groups (g) is at least two and the information of $g - 1$ groups are used as the training set while all individuals of the remaining group are used as the testing set. For example, in the context of genomic selection, when the plant breeder is interested in predicting these lines in another environment, the same (or different) lines were frequently evaluated in g environments or years, that represent the groups. Jarquín et al. (2017) denotes this type of CV strategy as CV1 in the context of plant breeding. Under this approach, the predictions are reported for each of the g groups because the scientist is interested in the prediction performance of each environment. Many times, the groups are the years under study and the aim is to predict the information of a complete year. However, when the groups are years and if we suspect that there is a considerable correlation between observations that are near in time. Therefore, it is

Fold 1	TRN	TST			
Fold 2	TRN		TST		
Fold 3	TRN			TST	
Fold 4	TRN				TST
	Year 1	Year 2	Year 3	Year 4	Year 5

Fig. 4.5 Schematic representation of time series data with 5 years, using the previous year for predicting the next year. However, in some practical applications, we are not interested in going too far back in time since the training and testing sets will be less related the farther you go

imperative to evaluate our statistical machine learning model for time series data on "future" observations. In this sense, the training sets are composed of the previous years to predict the subsequent year. This method can also be seen as a variation of k-fold CV, where the first folds are used for training the statistical machine learning model and the fold $(k + 1)$ is the corresponding testing set. The main difference in this CV method is that successive training sets are supersets of those that come before them. Also, it adds all surplus data to the first training partition, which is always used to train the model (see Fig. 4.5).

Figure 4.5 shows that under this type of CV, for time series data, the predictions are for $g - 1$ years; individuals from the first year are not predicted since a training set is not available.

4.3.7 Bootstrap Cross-validation

First, we will define the bootstrapping method to understand how it is used in the CV approach, which should then be straightforward. Bootstrapping is a type of resampling method where, for example, $B = 10$ samples of the same size are repeatedly drawn, with replacement, from a single original sample. Afterward, each of these B samples is used to estimate statistics (for example, the mean, variance, median, minimum, etc.) of a population, and the average of all the B sample estimates of the target statistic is reported as the final estimate. In the context of statistical machine learning, these samples are used to evaluate the prediction performance of the algorithm under study for unseen data. One important difference between this CV approach and all the procedures explained above is that now the training set has the same size (number of observations) as the original sample because the bootstrap method replaced some individuals more than once. According to Kuhn and Johnson (2013), as a result, some observations will be represented multiple times in the bootstrap sample, while others will not be selected at all; those observations not selected are referred to as the testing set, however, this CV strategy is quite different than the previously explained. Efron (1983) pointed out that the prediction performance of the bootstrap samples tends to have less uncertainty than the k-fold cross-validation since on average, 63.2% of the data points are represented (for training) at least once in any sample size. For this reason,

Fold	Training	Testing
1	I7, I4, I6, I9, I2, I3, I4, I4, I8, I6, I8, I7	I1, I5, I10, I11, I12
2	I2, I8, I5, I6, I1, I4, I5, I11, I1 1,I8,I10, I5	I3, I7, I9, I12
3	I5, I9, I11, I3, I10, I5, I7, I2, I 3, I11, I6, I9	I1, I4, I8, I12
4	I10, I12, I9, I7, I4, I3, I1, I9, I3, I2, I1, I6	I5, I8, I11
5	I2, I10, I5, I12, I3, I6, I3, I7, I6, I6, I5, I7	I1, I4, I8, I9

Fig. 4.6 Schematic representation of bootstrap cross-validation

this CV approach has a bias similar to implementing a $k =$ two fold cross-validation, and as the training set becomes smaller, the bias becomes more problematic. To understand this CV method, we provide a simple example of how the training and testing samples are constructed. If we have a sample with 12 individuals denoted as I1, I2, …, I12, we will select $B = 5$ bootstrap samples. Each bootstrap sample is obtained with replacement and the individuals that appear in each one correspond to the training sample; those that are not present will correspond to the testing set. Figure 4.6 provides the five bootstrap samples; each training sample has the same size as the original, however, only some individuals appear in each bootstrap sample, while those individuals that do not appear are included in the testing set. For example, in the first fold, the training bootstrap sample contains seven different individuals (I2, I3, I4, I6, I7, I8, and I9), while the testing set contains five individuals (I1, I5, I10, I11, and I12). It is important to point out that since the training sample has the same size as the original sample, some individuals in the training sample are repeated at least twice; in the first fold, the individual I4 are repeated three times, whereas I6, I7, and I8 are repeated twice. Finally, similar to other methods, the statistical machine learning model is trained with each training set; likewise, the prediction performance of the model is evaluated in each testing set. The average of these sample predictions is reported as the estimated testing error.

4.3.8 Incomplete Block Cross-validation

Incomplete block (IB) CV should be used when there are J treatments evaluated in I blocks and the same treatments are evaluated in all the blocks. The idea behind this

Table 4.1 TRN data set for $I = 3$ blocks and $J = 10$ treatments with 70% of data for training

Block	Treatments per block						
1	1	3	5	7	8	9	10
2	2	3	4	5	6	7	9
3	1	2	4	6	7	8	10

CV method is that some treatments should be present in some blocks but absent in others, whereas the same treatment should be present in at least one environment (block). The theory of incomplete block designs developed in the experimental designs statistical area can be used to construct the training set. For example, under a balanced incomplete block (BIB) design, the term incomplete means that all treatments in each block cannot be evaluated, whereas balanced means that each pair of treatments occur together λ times. The training set is constructed by first defining the % of individuals in the TRN set using the equation $sI = Jr = N_{\text{TRN}}$, where J represents the number of treatments under study, I represents the number of blocks under study, r denotes the number of repetitions of each treatment, and s denotes the treatments per block. For example, suppose that we had $J = 10$ treatments and $I = 3$ blocks (that is, 30 individuals), and we decided to use $N_{\text{TRN}} = 21$ (70%) of the total individuals in the TRN set. Therefore, the number of treatments by block can be obtained by solving $(sI = N_{\text{TRN}})$ for s, which results in $s = N_{\text{TRN}}/I$. This means that $s = 21/3 = 7$ treatments per block. Then, the corresponding elements for the training set can be obtained with the function find.BIB(10, 3, 7) of the package crossdes of the R statistical software. The numbers used in the function find.BIB() denote the treatments, the blocks, and the treatments per block, respectively. Finally, the treatments that make up the TRN set are shown in Table 4.1.

According to Table 4.1, it is clear that each treatment is present in two blocks and missing in one block. For example, in Block 1 the testing set includes treatments 2, 4, and 6; in Block 2, the testing set is composed of treatments 1, 8, and 10; and in Block 3, the testing set is composed of treatments 3, 5, and 9.

4.3.9 Random Cross-validation with Blocks

Random cross-validation with blocks was proposed by Lopez-Cruz et al. (2015) and belongs to the so-called replicated TRN-TST cross-validation that appears in the publication of Daetwyler et al. (2012), since some individuals can never be part of the training set. This algorithm, like the incomplete block cross-validation, is appropriate when we are interested in evaluating J lines in I blocks or environments and tries to mimic a prediction problem faced by breeders in incomplete field trials where lines are evaluated in some, but not all, target environments. The algorithm for constructing the TRN-TST sets is described by the following steps: Step 1. Calculate the total number of observations under study as $N = J \times I$; Step 2. Define the proportion of observations used for training and testing, that is, P_{TRN} and P_{TST}; Step 3. Calculate the size of the testing set $N_{\text{TST}} = N \times P_{\text{TST}}$; Step 4. Choose N_{TST} lines at

Table 4.2 TRN-TST data sets for $J = 10$ lines and $I = 3$ environments

Environment	Lines									
	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10
1	TRN	TST	TRN	TRN	TST	TRN	TRN	TST	TRN	TRN
2	TST	TRN	TRN	TRN	TRN	TRN	TST	TRN	TST	TRN
3	TRN	TRN	TST	TST	TRN	TRN	TRN	TRN	TRN	TST

random without replacement if $J \geq N_{TST}$, and with replacement otherwise; Step 5. Each chosen line will then be assigned to one of the I environments chosen at random without replacement; Step 5. All the selected lines and environments will form the training set, while the lines and environments that were not chosen will form the corresponding testing set; and Step 6. Steps 1–5 are repeated depending on the number of TRN-TST partitions required (Lopez-Cruz et al. 2015). This CV is called CV2 in Jarquín et al. (2017). Next, we assume that we have ten lines or treatments and three environments or blocks that will form the corresponding training testing sets for only one partition: Step 1. The total number of observations under study is $N = 10 \times 3 = 30$; Step 2. We define $P_{TRN} = 0.7$ and $P_{TST} = 0.3$; Step 3. The size of the testing set is $N_{TST} = 30 \times 0.3 = 9$; Step 4. Since $J = 10 \geq N_{TST} = 9$, we selected the following lines at random without replacement: L1, L2, L3, L4, L5, L6, L7, L8, L9, and L10; and Step 5. Each chosen line was assigned to one of the $I = 3$ environments randomly chosen without replacement, as shown in Table 4.2. It is important to point out that this CV strategy only differs from the incomplete block cross-validation in the way the lines are allocated to blocks.

4.3.10 Other Options and General Comments on Cross-validation

It is important to highlight that when the data set is considerably large, it is better to randomly split it into three parts: a training set, a validation set (or tuning set), and a testing set. The training set and testing set are used as explained before, while the validation (tuning set) set is used to estimate the prediction error for model selection, which is the process of estimating the performance of different models in order to choose the best one, or to evaluate the chosen statistical machine learning model with a range of values of tuning hyperparameters to select the combination of hyperparameters with the best prediction performance and then use these hyperparameters (or best model) to evaluate the prediction performance in the testing set (Fig. 4.7). It is important to point out that Fig. 4.7 shows only one random split of the data in terms of the training, testing, and validation sets.

For example, assume that our data set has 50,000 rows (observations) and that we have decided to use 5000 of them for the testing set and another 5000 for the validation set. This means that 40,000 rows are left for the training set. At this point, we train our statistical machine learning model with each component of the

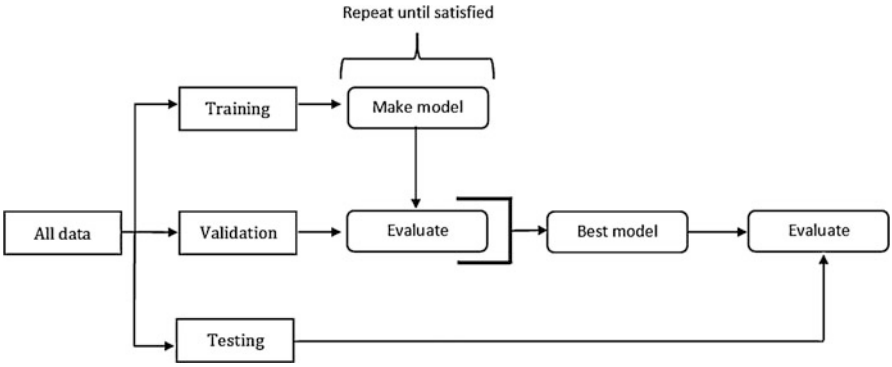

Fig. 4.7 Schematic representation of the training, validation (tuning), and testing sets proposed by Cook (2017)

grid of hyperparameters of the training set and evaluate the prediction performance on the validation set, as shown in the middle of Fig. 4.7. Finally, we will pick the best model (best subset of hyperparameters) in terms of prediction performance in the validation set and we are ready to evaluate the prediction performance in the testing set. Then, we will report the testing error on the testing set, as can be observed at the bottom of Fig. 4.7 (Cook 2017). Under this approach, the testing and validation sets have approximately the same size to guarantee a similar out-of-sample prediction performance. Since it is difficult to give general rules on how to choose the number of observations in each of the three parts, a typical number might be 50% for training, and 25% each for validation and testing (Hastie et al. 2008) or 70%, 15%, and 15% for training, validation, and testing, respectively. Another way to find the optimal setting of hyperparameters using the grid search, which is very common in deep learning, consists of picking values for each parameter from a finite set of options [e.g., number of epochs (100, 150, 200, 250, 300); batch sizes (25, 50, 75, 100, 125); number of layers (1, 2, 3, 4, 5); and types of activation functions (RELU, Sigmoid), ...] and training the statistical machine learning model with every permutation of hyperparameter choices using the training set. Then, the combination of hyperparameters with the best prediction performance on the validation set is chosen, and we report the prediction performance of the best selected model (set of hyperparameters) in the testing set (Buduma 2017). The aforementioned examples use the validation data set as a proxy measure of the accuracy during the hyperparameter optimization process. However, it can also be used as a proxy measure of the accuracy for model selection, and instead of using a grid of hyperparameters, we can use a set of different statistical machine learning models; here, the best model is chosen instead of the best combination of hyperparameters. It is important to understand that when you have more than one random partition (training-testing-validation), as shown in Fig. 4.8, the same process provided in Fig. 4.7 is followed, however, the average of all the partitions is reported as a measure of prediction performance. Also, if more precision is required in the estimated prediction performance, you can repeat the process given in Fig. 4.8

Partition 1	TRN						TST	VAL
Partition 2	TRN				TST		VAL	TRN
Partition 3	TRN		TST		VAL		TRN	
Partition 4	TST		VAL		TRN			
Partition 5	VAL		TRN					TST

Fig. 4.8 Five partitions of training-validation-testing

multiple times and report the average of all repetitions as a measure of prediction performance.

As mentioned above, this approach (Figs. 4.7 and 4.8) is used for a large data set however, in the case of a smaller data set, we suggest modifying this approach to avoid wasting too much training data in validation sets. This modification consists of performing an inner cross-validation approach since the training set is split into complementary subsets, and each model is trained against a different combination of these subsets, which is subsequently validated against the remaining parts. Once the model hyperparameters have been selected, a final model is trained with the whole training set (refitted) and the generalized prediction performance is measured on the testing set. This approach was applied by Montesinos-López et al. (2018a, b), who also split the original data into training and testing sets. Subsequently, each training set was split again and 80% of the data was used for training a grid of hyperparameters while the remaining 20% was used for validating (tuning) the prediction performance and selecting the best combination of hyperparameters with the best prediction performance. At this point, the deep learning algorithm was refitted with the whole training set, and with this they evaluated the out-of-sample prediction. They called the conventional training-testing partition outer cross-validation, while the split performed in each training set used for hyperparameter tuning was called inner cross-validation. It should be highlighted that there are no differences between outer and inner CV and training-validation-test. Finally, it should also be mentioned that any type of the cross-validation strategies mentioned in this section (random CV, *k*-fold CV, Bootstrap CV, IB CV, etc.,) can be used in both outer and inner CV, such as is the case with the five-fold CV.

4.4 Model Tuning

A hyperparameter is a parameter whose value is set before the learning process begins. Hyperparameters govern many aspects of the behavior of statistical machine learning models, such as their ability to learn features from data, the models' exhibited degree of generalizability in performance when presented with new data, as well as the time and memory cost of training the model, since different hyperparameters often result in models with significantly different performance. This means that tuning hyperparameter values is a critical aspect of the statistical machine learning training process and a key element for the quality of the resulting

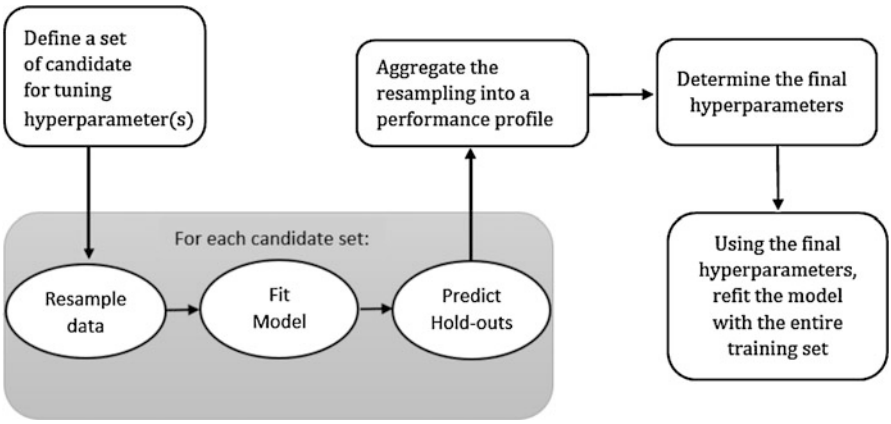

Fig. 4.9 Schematic representation of the tuning process proposed by Kuhn and Johnson (2013)

prediction accuracies. However, choosing appropriate hyperparameters is challenging (Montesinos-López et al. 2018a). Hyperparameter tuning finds the best version of a statistical machine learning model by running many training sets on the original data set using the algorithm and ranges of values of hyperparameters as specified. The hyperparameter values that provide the best performance in out-of-sample prediction evaluated by the chosen metric are then selected.

There are many ways of searching for the best hyperparameters. However, a general approach defines a set of candidate values for each hyperparameter. Each value of this set of candidate values is then applied with a resample of the training set of the chosen statistical machine learning method, where we aggregate all the hold-out predictions from which the best hyperparameters are chosen and refit the model with the entire set (Kuhn and Johnson 2013). A schematic representation of the tuning process proposed by Kuhn and Johnson (2013) is given in Fig. 4.9. It is important to highlight that this process should be performed correctly because when the same data are used for training and evaluating the prediction performance, the prediction performance obtained is extremely optimistic.

For example, suppose a breeder is interested in developing an algorithm to classify unseen plants as diseased or not diseased with an available training data set. The goal is to minimize the rate of misclassification or to maximize the percentage of cases correctly classified (PCCC). Also, assume that you are new to the world of statistical machine learning and that you only understand the k-nearest neighbor method. Since this algorithm depends only on the hyperparameter called the number of neighbors (k), the question is which value of k to choose in such a way that the prediction performance of this algorithm will be the best in the sample prediction of plants. To find the best value of the k hyperparameter, you must specify a range of values for k (for example, from 1 to 60 with increments of 1), then with a part of your training data set, called the training-inner (or tuning that corresponds to the training data in the inner loop) set, which is randomly selected. You proceed to evaluate the 60 values of k with the k-nearest neighbor method and evaluate the

prediction performance in the remaining part of the training set (validation set). Next, you select the value of k from this range of values that best predicts (according, for example, to the PCCC) out-of-sample data (validation set) and use this value to perform the prediction of the unseen plants not used for training the model (testing set). This is a widely adopted practice that consists of searching for the parameter (usually through brute force loops) that yields the best performance over a validation set. However, the process illustrated here is very simple because the k-nearest neighbor model only depends on a unique hyperparameter; however, there are other statistical machine learning algorithms (for example, deep learning methods) where the tuning process is required for a considerable amount of hyperparameters. For this reason, we encourage caution when choosing the statistical machine learning algorithm, since the amount of work required for performing the tuning process depends on the chosen method.

4.4.1 Why Is Model Tuning Important?

Tuning the hyperparameters of the models is a key element to optimize your statistical machine learning model to perform well in out-of-sample predictions. The tuning process is more an art than a science because there is no unique formal scientific procedure available in the literature. Nowadays, the tuning process is trial and error that consists of implementing the statistical machine learning model many times with different values of the hyperparameters and then comparing its performance on the validation set in order to determine which set of hyperparameters results in the most accurate model; for the final implementation, the set of hyperparameters of the best model is used. As mentioned above, for the k-nearest neighbor classifier, we need to choose the number of neighbors (k) using the tuning process to obtain the optimal prediction performance of this algorithm, while for conventional Ridge regression, the parameter lambda (λ) is obtained by tuning to improve the out-of-sample predictions. These two statistical machine learning algorithms that we just mentioned need only one hyperparameter; however, other statistical machine learning methods may require more hyperparameters, as exemplified by deep learning models that require at least three hyperparameters (number of neurons, number of hidden layers, type of activation function, batch size, etc.). After tuning the required hyperparameters, the statistical machine learning model learns the parameters from the data to be used for the final prediction of the testing set. The choice of hyperparameters significantly influences the time required to train and test a statistical machine learning model. Hyperparameters can be continuous or of the integer type; for this reason, there are mixed-type hyperparameter optimization methods.

4.4.2 Methods for Hyperparameter Tuning (Grid Search, Random Search, etc.)

Manual tuning of statistical machine learning models is of course possible, but relies heavily on the user's expertise and understanding of the underlying problem. Additionally, due to factors such as time-consuming model evaluations, nonlinear hyperparameter interactions in the case of large models that consist of tens or even hundreds of hyperparameters, manual tuning may not be feasible since it is equivalent to brute force. For this reason, the four most common approaches for hyperparameter tuning reported in the literature are (a) grid search, (b) random search, (c) Latin hypercube sampling, and (d) optimization (Koch et al. 2017).

In the *grid search* method, each hyperparameter of interest is discretized into a desired set of values to be studied where the models are trained and assessed for all combinations of the values across all hyperparameters (that is, a "grid"). Although fairly simple and straightforward to carry out, a grid search is appropriate when there are only a few values for a limited number of hyperparameters. However, although this is a comprehensive way of assessing different hyperparameter values, when there are many values for some or many hyperparameters, it quickly becomes quite costly due to the number of hyperparameters and the number of discrete levels of each. For example, in Ridge regression, this approach is implemented as follows: since λ is the hyperparameter to be tuned, we first propose, for example, a grid of 100 values for this hyperparameter from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$; then we divide the training set into five inner training sets and five inner testing (tuning) sets, where each of the 100 values of the grid is fitted using the inner training sets and the testing error is evaluated with the inner testing sets. Then we get the average predicted test error and pick one value out of the 100 values of the grid that produces the best prediction performance. Next, we refit the statistical machine learning method to the whole training set using the picked value of λ, and finally perform the predictions for the testing set using the learned parameters of the training set with the best picked value of λ. In all the models with one hyperparameter, it is practical to implement the *grid search* method, but for example, in deep learning models, which many times require six hyperparameters to be tuned, if only three values are used for each hyperparameter, there are $3^6 = 729$ combinations that need to be evaluated, quickly becoming computationally impracticable.

A *random search* differs from a *grid search* in that rather than providing a discrete set of values to explore each hyperparameter, we determine a statistical distribution for each hyperparameter from which values may be randomly sampled. This affords a much greater chance of finding effective values for each hyperparameter. While Latin hypercube sampling is similar to the previous method, it is a more structured approach (McKay 1992) since it is an experimental design in which samples are exactly uniform across each hyperparameter but random in combinations. These so-called low-discrepancy point sets attempt to ensure that points are approximately equidistant from one another in order to fill the space

efficiently. This sampling supports coverage across the entire range of each hyperparameter and is more likely to find good values of each hyperparameter.

The previous two methods for hyperparameter tuning are used to perform individual experiments by building models with various hyperparameter values and recording the model performance for each. Because each experiment is performed in isolation, this process is parallelized, but is unable to use the information from one experiment to improve the next experiment. Optimization methods, on the other hand, consist of sequential model-based optimization where the results of previous experiments are used to improve the sampling method of the next experiment. These methods are designed to make intelligent use of fewer evaluations and thus save on the overall computation time (Koch et al. 2017). Optimization algorithms that have been used in statistical machine learning generally for hyperparameter tuning include Broyden–Fletcher–Goldfarb–Shanno (BFGS) (Konen et al. 2011), covariance matrix adaptation evolution strategy (CMA-ES) (Konen et al. 2011), particle swarm (PS) (Renukadevi and Thangaraj 2014), tabu search (TS), genetic algorithms (GA) (Lorena and de Carvalho 2008), and more recently, surrogate-based Bayesian optimization (Dewancker et al. 2016). Also, recently the use of the response surface methodology has been explored for tuning hyperparameters in random forest models (Lujan-Moreno et al. 2018). However, the implementation of these optimization methods is not straightforward because it requires expensive computation; also, software development is required for implementing these algorithms automatically. There have been advances in this direction for some machine learning algorithms in the statistical analysis system (SAS), R and Python software (Koch et al. 2017). An additional challenge is the potential unpredictable computation expense of training and validating predictive models using different hyperparameter values. Finally, although it is challenging, the tuning process often leads to hyperparameter settings that are better than the default values, as it provides a heuristic validation of these settings, giving greater assurance that a model configuration with a higher accuracy has not been overlooked.

4.5 Metrics for the Evaluation of Prediction Performance

The quality of prediction performance of any statistical machine learning method in a given data set consists of evaluating how close the predicted values are to the true observed ones. In other words, the prediction performance quantifies the matching degree between the predicted response value for a given observation and the true response value for that observation (James et al. 2013). However, the metrics used for quantifying the prediction performance depend on the type of response variable under study; for this reason, we subsequently give the most popular metrics used for this goal for four types of response variables.

4.5.1 Quantitative Measures of Prediction Performance

Before implementing a statistical machine learning model, we assume that we have our training observation $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and we estimate f, as $\widehat{f}$, with the chosen statistical machine learning model. Then we can make predictions for each of the response values (y_i) with $\widehat{f}(x_i)$ and compute the predicted values for each of the n observations in the training set; with these values we can calculate the ***mean square error*** (MSE) for the training data set as $E = \frac{1}{n}\left(\sum_{i=1}^{n}\left(y_i - \widehat{f}(x_i)\right)\right)^2$; however, what we really want to predict are the values for unseen test observations that were not used to train the statistical machine learning model. Assuming that the unseen testing set is equal to $\{(x_{n+1}, y_{n+1}), (x_{n+2}, y_{n+2}), \ldots, (x_{n+T}, y_{n+T})\}$, the MSE for the testing data set should be calculated as

$$\mathrm{MSE}_{\mathrm{TST}} = \frac{1}{T}\sum_{i=n+1}^{n+T}\left(y_i - \widehat{f}(x_i)\right)^2, \tag{4.1}$$

where $\widehat{f}(x_i)$ is the prediction that $\widehat{f}$ gives to the ith observation. The $\mathrm{MSE}_{\mathrm{TST}}$ with a lower value will have better predictions, which means that the predicted values are very close to the true observed values. Also, the square root of $\mathrm{MSE}_{\mathrm{TST}}$ can be used as a measure of prediction performance and is called root mean square error (RMSE).

Pearson's correlation coefficient is a very popular measure of prediction performance in plant breeding and can be calculated as

$$r_{\mathrm{TST}} = \frac{\sum_{i=n+1}^{n+T}\left(\widehat{f}(x_i) - \overline{\widehat{f}(x_i)}\right)(y_i - \overline{y}_i)}{\sqrt{\sum_{i=n+1}^{n+T}\left(\widehat{f}(x_i) - \overline{\widehat{f}(x_i)}\right)^2}\sqrt{\sum_{i=n+1}^{n+T}(y_i - \overline{y}_i)^2}}, \tag{4.2}$$

where $\overline{\widehat{f}(x_i)}$ is the average of the T predictions that conform to the testing set, and $\overline{y}_i$ is the average of the T true observed values. In this case, the closer that the predictions are to 1, the better the implemented statistical machine learning model will perform. It is important to point out that Pearson's correlation is defined between -1 and 1. However, to be convinced that the observed and predicted values match, it is a common practice to perform a scatter plot of predicted versus observed (or vice versa) values and when the observed and predicted values follow a straight line (45° diagonal line) from the bottom left corner to the top right corner, this indicates a perfect match between the observed and predicted values. For this reason, Pearson's correlation as a metric should be complemented with the intercept and slope, since the slope and intercept describe the consistency and the model bias, respectively (Smith and Rose 1995; Mesple et al. 1996). To obtain the slope and intercept, the observed values (as y) versus the predicted values (as x) are regressed and in addition

to Pearson's correlation, these values should also be reported (slope and intercept). The expected intercept should be zero and the slope 1, if the correlation obtained between the observed and predicted values is high. It is important to avoid carrying out the regression in the opposite way, i.e., using predicted values as y's, and observed values as x's, since this leads to incorrect estimates of the slope and the y-intercept. This denotes that a spurious effect is added to the regression parameters when regressing predicted versus observed values and comparing them against the 1:1 line. The user should also remember that underestimation of the slope and overestimation of the y-intercept increase as Pearson's correlation values decrease. We strongly recommend that scientists evaluate their models by regressing observed versus predicted values and test the significance of slope $= 1$ and intercept $= 0$ (Piñeiro et al. 2008). Finally, it is important to recall that the square of Pearson's correlation can also be used as a metric for measuring prediction performance since it represents the proportion of the total variance explained by the regression model and is called the coefficient of determination denoted as R^2.

Next, we present the mean absolute error (MAE) metric that measures the difference between two continuous variables (observed and predicted). The MAE can be calculated with the following expression:

$$\mathrm{MAE}_{\mathrm{TST}} = \frac{1}{T}\sum_{i=n+1}^{n+T}\left|y_i - \widehat{f}(x_i)\right| \tag{4.3}$$

Below we present another metric used to evaluate the prediction performance of any statistical machine learning model; it was proposed by Kim and Kim (2016) and is called ***mean arctangent absolute percentage error*** (MAAPE) that is calculated as follows:

$$\mathrm{MAAPE}_{\mathrm{TST}} = \frac{\sum_{i=n+1}^{n+T}\arctan\left(\left|\frac{y_i - \widehat{f}(x_i)}{y_i}\right|\right)}{T} \tag{4.4}$$

Although MAAPE is finite when the response variable (i.e., $y_i = 0$) equals zero, since it has a satisfactory trigonometric representation. However, because MAAPE's value is expressed in radians, it is less intuitive, in addition to being scale-free. This metric is a modification of the mean absolute percentage error (MAPE) which is problematic because it is undefined when the response variable is equal to zero ($y_i = 0$). MAAPE is also asymmetric since division by zero is defined and is not a problem. It is important to point out that there are other metrics for measuring prediction accuracy for continuous data, but we only presented the most popular ones.

The distinction between the training and test MSE is important since we are not interested in how well the statistical machine learning method performs in the training data set, due to the fact that our main goal is to perform accurate predictions in the unseen test data. For example, a plant breeder may be interested in developing an algorithm to predict disease resistance of a plant in new environments based on

records that were collected in a set of environments. We can train the statistical machine learning method with the information collected in the set of environments, but the interest is not in how well the statistical machine learning method predicts in those previously collected environments. An environmental scientist can also be interested in predicting the average annual rainfall in a municipality in Mexico, using data from the last 20 years to train the model. Such measures could include sea surface temperature, time, the yearly rotation of the earth, among others. In this case, the scientist is really interested in predicting the average rainfall of the next 1 or 2 years, not in accurately predicting the years measured in the training set.

4.5.2 *Binary and Ordinal Measures of Prediction Performance*

The binary and ordinal response variables are very common in classification problems, where the goal is to predict which category something falls into. An example of a classification problem is analyzing financial data to determine if a client will be granted credit or not. Another example is analyzing breeding data to predict if an animal is at high risk for a certain disease or not. Below, we provide some popular metrics to evaluate the prediction performance of this type of data.

The first metric is called a ***confusion matrix***, which is a tool to visualize the performance of a statistical machine learning algorithm that is used in supervised learning for classifying categorical and binary data. Each column of the matrix represents the number of predictions in each class, while each row represents the instances in the real class. One of the benefits of confusion matrices is that they make it easy to determine whether the system is confusing classes. Table 4.3 shows a sample format of a confusion matrix of C classes.

With Eqs. (4.5–4.8) we can calculate the total number of false negatives (TFN), false positives (TFP), true negatives (TTN) for each class i, and the total true positives in the system, respectively:

$$\text{TFN}_i = \sum_{\substack{j=1 \\ j \neq i}}^{C} n_{ij} \tag{4.5}$$

Table 4.3 Confusion matrix with more than two classes

		Predicted values			
		Class 1	Class 2	...	Class C
Observed values	Class 1	n_{11}	n_{12}	...	n_{1C}
	Class 2	n_{21}	n_{22}	...	n_{2C}
	⋮	⋮	⋮	⋮	⋮
	Class C	n_{C1}	n_{C2}	...	n_{CC}

$$\mathrm{TFP}_i = \sum_{\substack{j=1 \\ j \neq i}}^{C} n_{ji} \tag{4.6}$$

$$\mathrm{TTN}_i = \sum_{\substack{j=1 \\ j \neq i}}^{C} \sum_{\substack{k=1 \\ k \neq i}}^{C} n_{ji} \tag{4.7}$$

$$\mathrm{TTP}_{\mathrm{all}} = \sum_{j=1}^{C} n_{jj} \tag{4.8}$$

Below, we define the sensitivity (S_e), precision (P), and specificity (S_p). The sensitivity indicates the ability of our statistical learning algorithm to determine the proportion of true positives that are correctly identified by the test. The precision is the proportion of correct classification of our statistical machine learning model and represents the proportion of cases correctly classified, while the specificity is the ability of our statistical machine learning model to classify the true negative cases, that is, the specificity is the proportion of true negatives that are correctly identified by the test. Under the "one-versus-all basis," where each category is compared with the composed information of the remaining categories, we provide the expressions for computing the generalized precision, sensitivity, and specificity for each class i:

$$P_i = \frac{\mathrm{TTP}_{\mathrm{all}}}{\mathrm{TTP}_{\mathrm{all}} + \mathrm{TFP}_i} \tag{4.9}$$

$$S_{ei} = \frac{\mathrm{TTP}_{\mathrm{all}}}{\mathrm{TTP}_{\mathrm{all}} + \mathrm{TFN}_i} \tag{4.10}$$

$$S_{pi} = \frac{\mathrm{TTN}_{\mathrm{all}}}{\mathrm{TTN}_{\mathrm{all}} + \mathrm{TFP}_i} \tag{4.11}$$

$$\mathrm{pCCC} = \frac{\mathrm{TTN}_{\mathrm{all}}}{\sum_{i=1}^{C} \sum_{j=1}^{C} n_{ij}} \tag{4.12}$$

The term pCCC denotes the ***proportion of cases correctly classified***, which is a measure of the overall accuracy, and when multiplied by 100, denotes the percentage of cases correctly classified. Many times, this is the only metric reported for measuring prediction performance in multi-class problems. However, PCCC alone is sometimes quite misleading as there may be a model with relatively "high" accuracy, but it predicts the "unimportant" class labels fairly accurately (e.g., "unknown bucket"). However, the model may be making all sorts of mistakes on the classes that are actually critical to the application. This problem is serious when in the input data the number of samples of different classes is very unbalanced. For

Table 4.4 Confusion matrix with two classes

		Predicted values		
		True	False	Sum
Observed values	True	tp	fn	tp + fn
	False	fp	tn	fp + tn
	Sum	tp + fp	fn + tn	n

tp denotes true positives, fp denotes false positives, fn denotes false negatives, tn denotes true negatives, and n denotes total number of individuals

example, if there are 990 samples of class 1 and only 10 of class 2, the classifier can easily have a bias toward class 1. If the classifier classifies all samples as class 1, its accuracy will be 99%. This does not mean that it is an appropriate classifier, as it had a 100% error when classifying the samples of class 2. For this reason, reporting this metric with those reported in Eqs. (4.9–4.11) is recommended in order to have a better picture of the prediction performance of any statistical machine learning method (Ratner 2017). Also, it is important to highlight that when the problem only has two classes, the confusion matrix is reduced to Table 4.4.

From Table 4.4 the PCCC is calculated as $\frac{tp+tn}{n}$, while the $S_e = \frac{tp}{tp+fn}$, $S_p = \frac{tn}{tn+fp}$, and $P = \frac{tp}{tp+fp}$. Also, when there are only two classes, González-Camacho et al. (2018) suggest calculating the ***Kappa coefficient*** (κ) or Cohen's Kappa, which is defined as

$$\kappa = \frac{P_0 - P_e}{1 - P_e},$$

where P_0 is the agreement between observed and predicted values and is computed by the PCCC described above for two classes; P_e is the probability of agreement calculated as $P_e = \frac{tp+fn}{n} \times \frac{tp+fp}{n} + \frac{fp+tn}{n} \times \frac{fn+tn}{n}$, where fp is the number of false positives, and fn is the number of false negatives (Table 4.4). This statistic can take on values between -1 and 1; a value of 0 means there is no agreement between the observed and predicted classes, while a value of 1 indicates perfect agreement between the model prediction and the observed classes. Negative values indicate that a prediction may be incorrect; however large negative values seldom occur when working with predictive models. Depending on the context, a Kappa value from 0.30 to 0.50 indicates reasonable agreement (Kuhn and Johnson 2013). The Kappa coefficient is appropriate when data are unbalanced, because it estimates the proportion of cases that were correctly identified by taking into account coincidences expected from chance alone (Fielding and Bell 1997). It is important to point out that this statistic was originally designed to assess the agreement between two raters (Cohen 1960).

Another popular metric for binary data is the ***Area Under the receiver operating characteristic*** Curve (AUC–ROC) and it ranks the positive predictions higher than the negative. The ROC curve is defined as a plot of 1 − specificity or false positive rate (FPR) as the x-axis versus its model sensitivity as the y-axis. For a given set of

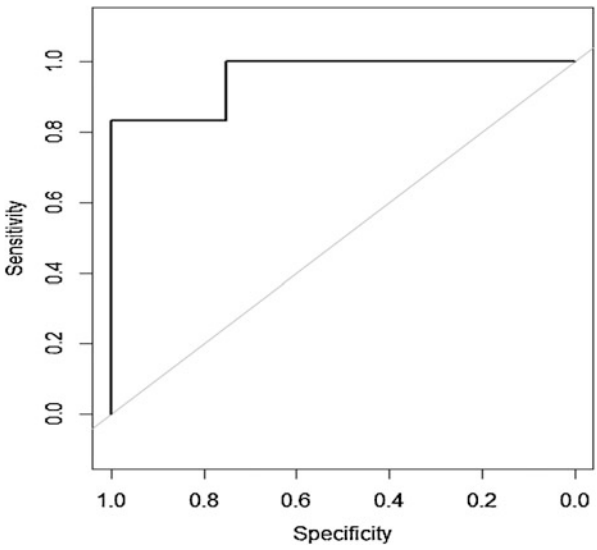

Fig. 4.10 ROC curve for the synthetic data

thresholds τ, it is an effective method for evaluating the quality or performance of diagnostic tests, and is widely used in statistical machine learning to evaluate the prediction performance of learning algorithms. Since the mathematical construction of this metric is not required in this book, we illustrate its calculation with one simple example. Assume that the observed (y), predicted probabilities (*pi*) and predicted values ($\hat{y}$) obtained after implementing a statistical machine learning model are $y = \{1, 0, 1, 1, 0, 0, 1, 1, 0, 1\}$, $pi = \{0.6, 0.55, 0.8, 0.78, 0.3, 0.42, 0.9, 0.45, 0.3, 0.88\}$, and $\widehat{y} = \{1, 1, 1, 1, 0, 0, 1, 0, 0, 1\}$; then, using the following R code, we can obtain many metrics for binary data (Fig. 4.10):

```
#########################Libraries required#########################
   library(caret)
   library(pROC)
   ##Observed (y), predicted probability (pi) and predicted values of
synthetic data##
   y=c(1, 0,1, 1, 0, 0, 1, 1, 0, 1)
   pi=c(0.6, 0.55, 0.8, 0.78, 0.3, 0.42, 0.9, 0.45, 0.3, 0.88)
   yhat=c(1, 1, 1, 1, 0, 0, 1, 0, 0, 1)
   xtab <- table(y,yhat)
   confusionMatrix(xtab)

   plot.roc(y,pi) #####This make the ROC curve plot

   Confusion Matrix and Statistics

     yhat
   y  0 1
    0 3 1
    1 1 5
```

```
               Accuracy : 0.8
                  95% CI : (0.4439, 0.9748)
    No Information Rate : 0.6
    P-Value [Acc > NIR] : 0.1673

                  Kappa : 0.5833
 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.7500
            Specificity : 0.8333
         Pos Pred Value : 0.7500
         Neg Pred Value : 0.8333
             Prevalence : 0.4000
         Detection Rate : 0.3000
   Detection Prevalence : 0.4000
      Balanced Accuracy : 0.7917

       'Positive' Class : 0
```

It is important to point out that when there are more than two classes, these metrics (accuracy, sensitivity, specificity, etc.) can be calculated on a "one-versus-all" basis that consists of using each class versus the pool of the remaining classes, as was illustrated for the confusion matrix with more than two classes (James et al. 2013).

When the statistical machine learning model discriminates correctly between the two groups, it produces a curve that coincides with the left and top sides of the plot. Under this scenario, the perfect model would have 100% sensitivity and specificity, and the ROC curve would be a single step between (0,0) and (0,1) and would remain constant from (0,1) to (1,1); this implies that the area under the ROC curve of the model would be 1. In general, the larger the area under the ROC curve, the better the model in terms of prediction performance. On the other hand, a completely useless statistical machine learning algorithm would give a straight line (45° diagonal line) from the bottom left corner to the top right corner of the plot. Different statistical machine learning models or the same model with different training sets or hyperparameters can be compared by superimposing their ROC curves in the same graph. In practice, most of the time the values in the two groups overlap, so the curve often lies between these extremes.

From this ROC curve, we can obtain a global assessment of the prediction performance of the statistical machine learning method by measuring the area under the receiver operating characteristic curve. This area is equal to the probability that a random individual (person) of the sample with the presence of the target has a higher value of the measurement than a random individual without the target.

When a statistical machine learning model is unable to discriminate between the positive and negative classes, this means that it has low discriminatory power. Therefore, only in statistical machine learning models that had good discriminatory power we can be confident of the predictions they provide and furthermore those models that provide a curve that lies considerably above the curve will be better.

Matthews correlation coefficient (MCC). Introduced in 1975 by Brian Matthews (1975) and regarded by many scientists as the most informative score that connects all four measures in a confusion matrix, the Matthews Correlation Coefficient is typically used in statistical machine learning to measure the quality of binary classifications and it is particularly useful when there is a significant imbalance in class sizes (data). MCC is calculated according to the following expression:

$$\text{MCC} = \frac{\text{tp} \times \text{tn} - \text{fp} \times \text{fn}}{\sqrt{(\text{tp} + \text{fp}) \times (\text{tp} + \text{fn}) \times (\text{tn} + \text{fp}) \times (\text{tn} + \text{fn})}} \tag{4.13}$$

If any of the denominator terms equals zero in (4.13) it will be set to 1 and MCC becomes zero, which has been shown to be the correct limiting value. It returns a value between -1 and 1, where 1 means a perfect prediction, 0 means no better than random and -1 means a total disagreement between predicted and observed values.

Next, we present the ***Brier score*** (Brier 1950) for categorical or binary data that can be computed as

$$\text{BS} = T^{-1} \sum_{i=n+1}^{n+T} \sum_{c=1}^{C} (\widehat{\pi}_{ic} - d_{ic})^2, \tag{4.14}$$

where $\widehat{\pi}_i$ denotes the estimated probabilities (predictive distribution) derived from the estimated model for observation i and d_{ic} takes a value of 1 if the categorical response observed for individual i falls into category c; otherwise, $d_{ic} = 0$. The range of BS in Eq. (4.14) for categorical data is between 0 and 2. For this reason, we suggest dividing by 2, that is, BS/2, to obtain the Brier score bound between 0 and 1; lower scores imply better predictions (Montesinos-López et al. 2015a, b).

Finally, we describe the use of negative log-likelihood (MLL) to evaluate the prediction performance. This metric has the characteristic that better forecasts have lower values and for this reason, it is analogous to the MSE. For categorical data,

$$\text{MLL} = -\frac{1}{T}\left[\sum_{i=n+1}^{n+T} \sum_{c=1}^{C} 1\{y_i = k\} \log(\widehat{\pi}_{ic})\right],$$

where $1\{y^{(i)} = k\}$ is an indicator variable taken the value of 1 when the ith observation is assigned to category c, for $c = 1, 2, \ldots, C$ takes place in the ith observation. When the data are binary, the MLL is reduced to $\text{MLL} = -\frac{1}{T}\left[\sum_{i=n+1}^{n+T}[y_i \log(\widehat{\pi}_i) + (1 - y_i) \log(1 - \widehat{\pi}_i)]\right]$. Following, we provide some advantages of using the MML as a measure of prediction performance: (a) it has a simple definition that, from a purely intuitive point of view, seems to be a reasonable basis on which to compare forecasts; (b) it is mathematically optimal in the sense that estimates of parameters of calibration models fitted by maximizing the likelihood are usually the most accurate possible estimates (see Cassella and Berger 2002); (c) it is a generalization to probabilistic forecasts of the most commonly used skill score for single forecasts; (d) the properties of the likelihood have been studied

at great length over the last 90 years, and as such is well understood; (e) it is both a measure of resolution and reliability; (f) likelihood can be used for both calibration and assessment: this creates consistency between these two operations; (g) use of the likelihood also creates consistency with other statistical modeling activities, since most other statistical modeling uses the likelihood, which is important in cases where the use of forecasts is simply a small part of a larger statistical modeling effort, as is the case of our particular business; (h) likelihood can be used for all types of response variables; and (i) likelihood can be used to compare multiple leads, multiple variables, and multiple locations at the same time in a sensible way with a single score even when these leads, variables, and locations are cross-correlated.

4.5.3 Count Measures of Prediction Performance

Spearman's correlation and the MML are recommended to measure the prediction performance for count data.

For the application of the ***Spearman's correlation***, the formula given in Eq. (4.2) for Pearson's correlation can be used; however, instead of using the observed and predicted values directly, these are replaced by their corresponding ranks. For example, assuming that the observed and predicted values are $y = \{15, 9, 12, 27, 6, 3, 36, 15, 21, 30\}$ and $\widehat{y} = \{20, 17, 24, 25, 3, 3, 34, 22, 21, 33\}$, we can thus show how to get the rank for the observed values: $\text{rango}_y = \{5, 3, 4, 8, 2, 1, 10, 6, 7, 9\}$. However, in this vector, observations 1 and 8 are the same, and as such, their positions are added and divided by two, that is, $\frac{5+6}{2} = 5.5$. Therefore, the final rank for the observed values is $\text{rango}_y = \{5.5,3, 4, 8, 2, 1, 10,5.5,7, 9\}$. Now the range for the predicted values is $\text{rango}_{\widehat{y}} = \{4, 3, 7, 8, 1, 2, 10, 6, 5, 9\}$, but again, since values 5 and 6 are the same, we add their ranges, and as this is repeated twice, it is divided by two and we get $\frac{1+2}{2} = 1.5$. Therefore, the final range of the predicted values is $\text{rango}_{\widehat{y}} = \{4, 3, 7, 8, 1.5, 1.5, 10, 6, 5, 9\}$. Finally, to obtain the Spearman correlation, we used the expression given in Eq. (4.2) for Pearson's correlation, and instead of using the original observed and predicted values, we used rango_y and $\text{rango}_{\widehat{y}}$. The interpretation of this metric is equal to that of the Pearson correlation, that is, when it is closer to 1, the prediction performance of the implemented statistical learning method is better. It should be noted that when the number of repeated values in the observed and predicted values is greater than two, the adjusted range is the sum of the repeated ranges divided by the number of repeated values; this new range is then given to the repeated values. In this case, it is also important to regress the observed versus the predicted values to obtain the intercept and slope using the ranges of the observed and predicted values.

It is also possible to use the MLL criteria to assess the prediction performance for count data; however, the new expression is now based on minus the log-likelihood of a Poisson distribution, which is equal to

$$\text{MLL} = \frac{1}{T}\sum_{i=n+1}^{n+T}\left[-\widehat{f}(x_i) + y_i \log\left(\widehat{f}(x_i)\right)\right]$$

Again, when the values of MLL are lower, the observed and predicted values are closer to one another.

References

Brier GW (1950) Verification of forecasts expressed in terms of probability. Mon Weather Rev 78: 1–3

Buduma M (2017) Fundamentals of deep learning, 1st edn. O'Reilly, Sabastopol, CA

Burger SV (2018) Introduction to machine learning with R. Rigorous mathematical analysis, 1st edn. O'Reilly, Sabastopol, CA

Cassella G, Berger RL (2002) Statistical inference. Duxbury, Belmont, CA

Cohen J (1960) A coefficient of agreement for national data. Educ Psychol Meas 20:37–46

Cook D (2017) Practical machine learning with H_2O. O'Reilly Media, Inc, Sabastopol, CA

Daetwyler HD, Calus MPL, Pong-Wong R, de los Campos G, Hickey JM (2012) Genomic prediction in animals and plants: simulation of data, validation, reporting and benchmarking. Genetics 193:347–365

Dewancker I, McCourt M, Clark S, Hayes P, Johnson A, Ke G (2016) A stratified analysis of Bayesian optimization methods. arXiv:1603.09441v1

Efron B (1983) Estimating the error rate of a prediction rule: improvement on cross-validation. J Am Stat Assoc 78(382):316–331

Fielding AH, Bell JF (1997) A review of methods for the assessment of prediction errors in conservation presence/absence models. Environ Conserv 24:38–49. https://doi.org/10.1017/S0376892997000088

González-Camacho JM, Ornella L, Pérez-Rodríguez P, Gianola D, Dreisigacker S, Crossa J (2018) Applications of machine learning methods to genomic selection in breeding wheat for rust resistance. Plant Genome 11(2):1–15. https://doi.org/10.3835/plantgenome2017.11.0104

Hagerty MR, Srinivasan S (1991) Comparing the predictive powers of alternative multiple regression models. Psychometrika 56:77–85. MR1115296

Hastie T, Tibshirani R, Friedman J (2008) The elements of statistical learning: data mining, inference, and prediction, Springer series in statistics, 2nd edn. Springer, New York

James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning: with applications in R. Springer, New York

Jarquín D, Lemes da Silva C, Gaynor RC, Poland J, Fritz AR et al (2017) Increasing genomic-enabled prediction accuracy by modeling genotype × environment interactions in Kansas wheat. Plant Genome 10(2):1–15. https://doi.org/10.3835/plantgenome2016.12.0130

Kim S, Kim H (2016) A new metric of absolute percentage error for intermittent demand forecasts. Int J Forecast 32(3):669–679

Koch P, Wujek B, Golovidov O, Gardner S (2017) Automated hyperparameter tuning for effective machine learning. In: Proceedings of the SAS global forum 2017 conference. SAS Institute Inc, Cary, NC. http://support.sas.com/resources/papers/proceedings17/SAS514-2017.pdf

Konen W, Koch P, Flasch O, Bartz-Beielstein T, Friese M, Naujoks B (2011) Tuned data mining: a benchmark study on different tuners. In: Proceedings of the 13th annual conference on genetic and evolutionary computation (GECCO-2011). SIGEVO/ACM, New York

Kuhn M, Johnson K (2013) Applied predictive modeling. Springer, New York

Lopez-Cruz M, Crossa J, Bonnett D, Dreisigacker S, Poland J, Jannink J-L, Singh RP, Autrique E, de los Campos, G. (2015) Increased prediction accuracy in wheat breeding trials using a marker × environment interaction genomic selection method. G3 5(4):569–582

Lorena AC, de Carvalho ACPLF (2008) Evolutionary tuning of SVM parameter values in multiclass problems. Neurocomputing 71:3326–3334
Lujan-Moreno GA, Howard PR, Rojas OG, Montgomery DC (2018) Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study. Expert Syst Appl 109:195–205
Matthews BW (1975) Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim Biophys Acta Protein Struct 405:442–451
McKay MD (1992) Latin hypercube sampling as a tool in uncertainty analysis of computer models. In: Swain JJ, Goldsman D, Crain RC, Wilson JR (eds) Proceedings of the 24th conference on winter simulation (WSC 1992). ACM, New York, pp 557–564
Mesple F, Troussellier M, Casellas C, Legendre P (1996) Evaluation of simple statistical criteria to qualify a simulation. Ecol Model 88:9–18
Montesinos-López OA, Montesinos-López A, Pérez-Rodríguez P, de los Campos G, Eskridge KM, Crossa J (2015a) Threshold models for genome-enabled prediction of ordinal categorical traits in plant breeding. G3 5(1):291–300
Montesinos-López OA, Montesinos-López A, Crossa J, Burgueño J, Eskridge K (2015b) Genomic-enabled prediction of ordinal data with Bayesian logistic ordinal regression. G3 5(10):2113–2126. https://doi.org/10.1534/g3.115.021154
Montesinos-López A, Montesinos-López OA, Gianola D, Crossa J, Hernández-Suárez CM (2018a) Multi-environment genomic prediction of plant traits using deep learners with a dense architecture. G3 8(12):3813–3828. https://doi.org/10.1534/g3.118.200740
Montesinos-López OA, Montesinos-López A, Crossa J, Gianola D, Hernández-Suárez CM et al (2018b) Multi-trait, multi-environment deep learning modeling for genomic-enabled prediction of plant traits. G3 8(12):3829–3840. https://doi.org/10.1534/g3.118.200728
Piñeiro G, Perelman S, Guerschman JP, Paruelo JM (2008) How to evaluate models: observed vs. predicted or predicted vs. observed? Ecol Model 216:316–322
Ratner B (2017) Statistical and machine-learning data mining. Techniques for better predictive modelling and analysis of big data, 3rd edn. CRC Press Taylor & Francis Group, Boca Raton, FL
Renukadevi NT, Thangaraj P (2014) Performance analysis of optimization techniques for medical image retrieval. J Theor Appl Inf Technol 59:390–399
Shalev-Shwartz S, Ben-David S (2014) Understanding machine learning from theory to algorithms. Cambridge University press, New York
Shmueli G (2010) To explain or to predict? Stat Sci 25(3):289–310
Smith EP, Rose KA (1995) Model goodness-of-fit analysis using regression and related techniques. Ecol Model 77:49–64
Wu S, Harris T, Mcauley K (2007) The use of simplified or misspecified models: linear case. Canad J Chem Eng 85:386–398

Chapter 5
Linear Mixed Models

5.1 General of Linear Mixed Models

Linear mixed models (LMM) are flexible extensions of linear models in which fixed and random effects enter linearly into the model. This is useful in many disciplines to model repeated, longitudinal, or clustered observations, in which random effects are introduced to help capture correlation or/and random variation among observations in the same group of individuals. Random effects are random values associated with levels of a random factor, and often represent random deviations from the population mean and linear relationships described by fixed effects (Pinheiro and Bates 2000; West et al. 2014).

The first formulation of a linear mixed model was applied in the field of astronomy to analyze repeated telescopic observations made at various hourly intervals over a range of nights (West et al. 2014). The mixed model approach is often called by various names, depending on the discipline in which it is applied. For example, in the social sciences, this approach is known as a multilevel or hierarchical model that is often used to flexibly measure the different levels of grouping present in the data structure (e.g., an impact evaluation of a new teaching method, survey of job satisfaction, education applications, etc.) (Goldstein 2011; Speelman et al. 2018; Finch et al. 2019).

Other application areas can be found in medicine (health care research, Leyland and Goldstein 2001; Brown and Prescott 2014), agriculture, ecology, industry, and animal science, where this model is often referred to as the random effects or mixed-effects model (Pinheiro and Bates 2000; Raudenbush and Bryk 2002; Meeker et al. 2011; Zuur et al. 2009). Specifically, now there is an increasing number of applications of this model in genomic selection for plant and animal breeding, where molecular markers obtained by genotyping-by-sequencing or other technologies are used to predict breeding values for non-phenotyped lines to select candidate lines prior to phenotypic evaluation (Meuwissen et al. 2001; Poland et al. 2012; Cabrera-Bosquet et al. 2012; Araus and Cairns 2014; Crossa et al. 2017;

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_5

Covarrubias-Pazaran et al. 2018; Wang et al. 2018; Cappa et al. 2019). However, the use of this model in animal science can be traced back to Henderson (1950).

The general univariate linear mixed model (Harville 1977) is provided by the formula

$$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{Zb} + \boldsymbol{\epsilon}, \tag{5.1}$$

where $\boldsymbol{Y}$ is the $n \times 1$ random response vector, $\boldsymbol{X}$ is the $n \times (p + 1)$ design matrix for the fixed effects, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^{\mathrm{T}}$ is the $(p + 1) \times 1$ coefficient vector of fixed effects, $\boldsymbol{b}$ is a $q \times 1$ vector of random effects, $\boldsymbol{Z}$ is the associate matrix design for the random effects, and $\boldsymbol{\epsilon}$ is the $n \times 1$ vector of random errors. It assumes that $\boldsymbol{\epsilon}$ is a random vector with a mean vector of $\mathbf{0}$ and a variance–covariance matrix $\boldsymbol{R}$, $\boldsymbol{b}$ is a random vector with a mean of $\mathbf{0}$ and variance–covariance matrix $\boldsymbol{D}$, and a null variance–covariance matrix between $\boldsymbol{\epsilon}$ and $\boldsymbol{b}$, $\mathrm{Cov}(\boldsymbol{\epsilon}, \boldsymbol{b}) = \mathbf{0}_{n \times q}$. In genomic applications, $\boldsymbol{b}$ often includes the genotypic effects and genotype × environment interaction effects, while $\boldsymbol{X}$ may contain information about environment covariates and other related information.

Note that under this model, $E(\boldsymbol{Y}) = \boldsymbol{X\beta}$ and the variance–covariance matrix of the response vector is $\mathrm{Var}(\boldsymbol{Y}) = \boldsymbol{ZDZ}^{\mathrm{T}} + \boldsymbol{R}$.

5.2 Estimation of the Linear Mixed Model

5.2.1 Maximum Likelihood Estimation

One method typically used for the estimation of the parameters of the LMM is the maximum likelihood approach. For the estimation under an LMM, the random errors and the random effects components are needed. Assuming that $\boldsymbol{\epsilon} \sim N_n(\mathbf{0}, \boldsymbol{R})$, and $\boldsymbol{b} \sim N_q(\mathbf{0}, \boldsymbol{D})$, with $\boldsymbol{R}$ and $\boldsymbol{D}$ positive semi-defined matrices, the marginal distribution of the response vector $\boldsymbol{Y}$ is $N_n(\boldsymbol{X\beta}, \boldsymbol{ZDZ}^{\mathrm{T}} + \boldsymbol{R})$, and so the likelihood of the parameters is given by

$$L(\boldsymbol{\beta}, \boldsymbol{D}, \boldsymbol{R}; \boldsymbol{y}) = \frac{|\boldsymbol{V}|^{-\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \exp\left[-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{X\beta})^{\mathrm{T}} \boldsymbol{V}^{-1} (\boldsymbol{y} - \boldsymbol{X\beta})\right], \tag{5.2}$$

where $\boldsymbol{V} = \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{DZ} + \boldsymbol{R}$ is the marginal variance of $\boldsymbol{Y}$.

The maximum likelihood estimators (MLE) of the parameters, $\boldsymbol{\beta}$, $\boldsymbol{D}$, and $\boldsymbol{R}$, are the values that maximize the likelihood function (5.2) (Searle et al. 2006; Stroup 2012), but due to the fact that no explicit formulas to estimate these parameters exist, numerical methods such as Newton–Raphson and Fisher Scoring are used. See details for implementing these methods in Jennrich and Sampson (1976) for the case where: $\boldsymbol{D}$ is a block diagonal with a submatrix in each diagonal of the form $\sigma_j^2 \boldsymbol{A}_j$, where $\boldsymbol{A}_j$ is a known matrix and σ_j^2 is the variance component parameter to be

estimated in this case; while for $\boldsymbol{R} = \sigma^2\boldsymbol{C}$, where $\boldsymbol{C}$ is a known matrix, only σ^2 should be estimated. For a more general explanation, see Jennrich and Schluchter (1986), and for an improvement of the algorithms proposed by these authors, consult Lindstrom and Bates (1988).

Another numerical method that can be used to obtain the MLE is the expected maximization (EM) algorithm, which is conceptually a simple algorithm for parameter estimation in this model (Laird and Ware 1982). This algorithm is an iterative numerical method to obtain maximum likelihood in the context of missing or hidden data (Borman 2004). The EM algorithm is described for the case where $\boldsymbol{R} = \sigma^2\boldsymbol{I}_n$, and where $\boldsymbol{I}_n$ is the identity matrix of dimension n. This algorithm, for some specific variance–covariance matrices of random effects, as described and used below, can be implemented using the sommer R package (Covarrubias-Pazaran 2016, 2018), which provides two additional algorithms available to obtain the MLE of the parameters in this same model.

5.2.1.1 EM Algorithm

The likelihood for complete data, $\boldsymbol{y}$ and $\boldsymbol{b}$, is given by

$$\begin{aligned} f_{\boldsymbol{Y},\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{b}) &= f_{\boldsymbol{Y}|\boldsymbol{b}}(\boldsymbol{y}|\boldsymbol{b}) f_{\boldsymbol{b}}(\boldsymbol{b}) \\ &= \frac{|\boldsymbol{D}|^{-\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\boldsymbol{b}) - \frac{1}{2}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{b}\right] \end{aligned}$$

As such, the log-likelihood for the complete data, $\boldsymbol{y}$ and $\boldsymbol{b}$, is given by

$$\begin{aligned} \ell_c(\boldsymbol{\beta},\boldsymbol{\theta};\boldsymbol{y},\boldsymbol{b}) &= \log\left[\, f_{\boldsymbol{Y},\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{b})\right] \\ &= -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\boldsymbol{b}) \\ &\quad -\frac{1}{2}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{b} - \frac{1}{2}\log\left(|\boldsymbol{D}|\right), \end{aligned}$$

where $\boldsymbol{\theta}$ is the vector parameter that defines the variance–covariance matrix of the random effects ($\boldsymbol{D}$) and the random vector ($\boldsymbol{R}$). Some specific examples are given below.

E Step

Because $E(\boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{u}) = \mathrm{tr}[\boldsymbol{A}\mathrm{Var}(\boldsymbol{u})] + E(\boldsymbol{u})^{\mathrm{T}}\boldsymbol{A}E(\boldsymbol{u})$ and $\boldsymbol{b} \mid \boldsymbol{Y} = \boldsymbol{y} \sim N_q\left(\widetilde{\boldsymbol{b}}, \widetilde{\boldsymbol{D}}\right)$ (see Appendix 1), given the current values of the parameters $\boldsymbol{\beta}_{(t)}$ and $\boldsymbol{\theta}_{(t)}$, the conditional expected value of the complete likelihood, $\ell_c(\boldsymbol{\beta},\boldsymbol{\theta};\boldsymbol{y},\boldsymbol{b})$, is given by [Step (E)]:

$$
\begin{aligned}
Q\left(\boldsymbol{\beta},\boldsymbol{\theta}|\boldsymbol{\beta}_{(t)},\boldsymbol{\theta}_{(t)},\right) &= E_{\boldsymbol{b}|\boldsymbol{y}}[\ell_c(\boldsymbol{\beta},\boldsymbol{b},\boldsymbol{\theta};\boldsymbol{y})] \\
&= -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{Z}\widetilde{\boldsymbol{D}}_{(t)}\boldsymbol{Z}^{\mathrm{T}}\right) - \frac{1}{2\sigma^2}\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right) \\
&\quad -\frac{1}{2}\mathrm{tr}\left(\boldsymbol{D}^{-1}\widetilde{\boldsymbol{D}}_{(t)}\right) - \frac{1}{2}\widetilde{\boldsymbol{b}}_{(t)}^{\mathrm{T}}\boldsymbol{D}^{-1}\widetilde{\boldsymbol{b}}_{(t)} - \frac{1}{2}\log\left(|\boldsymbol{D}|\right) \\
&= -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left[\mathrm{tr}\left(\boldsymbol{Z}\widetilde{\boldsymbol{D}}_{(t)}\boldsymbol{Z}^{\mathrm{T}}\right) + \left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)\right] \\
&\quad -\frac{1}{2}\left\{\mathrm{tr}\left[\left(\widetilde{\boldsymbol{D}}_{(t)}+\widetilde{\boldsymbol{b}}_{(t)}\widetilde{\boldsymbol{b}}_{(t)}^{\mathrm{T}}\right)\boldsymbol{D}^{-1}\right] + \log\left(|\boldsymbol{D}|\right)\right\}
\end{aligned}
$$

where $\widetilde{\boldsymbol{D}}_{(t)} = \left(\boldsymbol{D}_{(t)}^{-1} + \sigma_{(t)}^{-2}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\right)^{-1}$, $\widetilde{\boldsymbol{b}}_{(t)} = \sigma_{(t)}^{-2}\widetilde{\boldsymbol{D}}_{(t)}\boldsymbol{Z}^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}_{(t)}\right)$, and $\boldsymbol{\beta}_{(t)}$, $\sigma_{(t)}^2$, and $\boldsymbol{D}_{(t+1)}$ are the current values of $\boldsymbol{\beta}$, σ^2, and $\boldsymbol{D}$, respectively.

M Step

The second step of the EM algorithm is the M step, which consists of updating the parameters by maximizing the conditional expected value of the complete likelihood. First, we can observe that for any value of $\boldsymbol{\theta}$, the value of $\boldsymbol{\beta}$ that maximizes $Q(\boldsymbol{\beta},\boldsymbol{\theta}|\boldsymbol{\beta}_{(t)},\boldsymbol{\theta}_{(t)})$ is given by

$$
\boldsymbol{\beta}_{(t+1)} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)
$$

which does not depend on the chosen values of $\boldsymbol{\theta}$, but rather specifically on the chosen values of σ^2 and $\boldsymbol{D}$. Then by equating to zero, the derivative of $Q(\boldsymbol{\beta}_{(t+1)},\boldsymbol{\theta}|\boldsymbol{\beta}_{(t)},\boldsymbol{\theta}_{(t)})$ with respect to σ^2, and solving for σ^2, we can obtain that the value of σ^2 that maximizes $Q(\boldsymbol{\beta}_{(t+1)},\boldsymbol{\theta}|\boldsymbol{\beta}_{(t)},\boldsymbol{\theta}_{(t)})$, for $\boldsymbol{D}$ fixed, is given by

$$
\sigma_{(t+1)}^2 = \frac{1}{n}\left[\mathrm{tr}\left(\boldsymbol{Z}\widetilde{\boldsymbol{D}}_{(t)}\boldsymbol{Z}^{\mathrm{T}}\right) + \left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}_{(t+1)}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)^{\mathrm{T}}\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}_{(t+1)}-\boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)\right]
$$

which is independent of the value of $\boldsymbol{D}$. Now, according to result 4.10 in Johnson and Wichern (2002), the value of $\boldsymbol{D}$ that maximizes $Q(\boldsymbol{\beta},\boldsymbol{\theta}|\boldsymbol{\beta}_{(t)},\boldsymbol{\theta}_{(t)})$ is given by

$$
\boldsymbol{D}_{(t+1)} = \widetilde{\boldsymbol{D}}_{(t)} + \widetilde{\boldsymbol{b}}_{(t)}\widetilde{\boldsymbol{b}}_{(t)}^{\mathrm{T}}
$$

and does not depend on $\boldsymbol{\beta}$ and σ^2. So, by joining the above optimization, we have the M step that consists of updating the parameters $\boldsymbol{\beta}$, σ^2, and $\mathbf{D}$, with $\boldsymbol{\beta}_{(t+1)}$, $\sigma_{(t+1)}^2$, and $\boldsymbol{D}_{(t+1)}$, respectively. At this point, we can observe that the current value of the parameters $\boldsymbol{\beta}_{(t)}$, $\sigma_{(t)}^2$, and $\boldsymbol{D}_{(t)}$ are used in the computation of $\widetilde{\boldsymbol{b}}$ and $\widetilde{\boldsymbol{D}}$.

In the case of $\boldsymbol{D} = \sigma_g^2\boldsymbol{A}$, where $\boldsymbol{A}$ is a known matrix (the case in some genomic prediction models, where $\boldsymbol{A}$ corresponds to the genomic relationship matrix, the pedigree matrix, or the environmental matrix), the unique variance parameters to

estimate are σ_g^2 and σ^2, that is, $\left(\boldsymbol{\theta} = \left(\sigma_g^2, \sigma^2\right)\right)$. In the same fashion, very often in genomic applications but also in a more general setting, $\boldsymbol{D} = \text{Diag}\left(\sigma_1^2 \boldsymbol{A}_1, \ldots, \sigma_K^2 \boldsymbol{A}_K\right)$, where $\boldsymbol{A}_k$ represents a known variance–covariance matrix (or correlation) structure (genomic relationship matrix, pedigree relationship matrix, etc., Burgueño et al. 2012) between the different random effects included. In this context, the variance component parameters to estimate are $\sigma_k^2, k = 1, \ldots, K$, and σ^2 $\left(\boldsymbol{\theta} = \left(\sigma_1^2, \ldots, \sigma_K^2, \sigma^2\right)\right)$, where $\boldsymbol{A}_k$, $k = 1, \ldots, K$, are positive defined known matrices of dimensions $q_k \times q_k$, $k = 1, \ldots, K$, such that $\sum_{k=1}^{K} q_k = q$, the E step can be reduced (see Appendix 2) to

$$Q\left(\boldsymbol{\beta}, \boldsymbol{\theta} | \boldsymbol{\beta}_{(t)}, \boldsymbol{\theta}_{(t)}\right) = -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left[\text{tr}\left(\boldsymbol{Z}\widetilde{\boldsymbol{D}}_{(t)}\boldsymbol{Z}^{\text{T}}\right) + \left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)^{\text{T}}\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}\widetilde{\boldsymbol{b}}_{(t)}\right)\right]$$
$$-\frac{1}{2}\sum_{k=1}^{K}\left\{\frac{\sigma_{k(t)}^2}{\sigma_k^2}\left[q_k - \sigma_{k(t)}^2 \text{tr}\left(\boldsymbol{A}_k \boldsymbol{Z}_k^{\text{T}} \boldsymbol{V}_{(t)}^{-1} \boldsymbol{Z}_k\right) + \sigma_{k(t)}^{-2} \widetilde{\boldsymbol{b}}_{k(t)}^{\text{T}} \boldsymbol{A}_k^{-1} \widetilde{\boldsymbol{b}}_{k(t)}\right] + q_k \log\left(\sigma_k^2\right) + \log\left(|\boldsymbol{A}_k|\right)\right\},$$

where $\boldsymbol{V}_{(t)}$ is the marginal matrix of variance–covariance of the response vector in the current value of the parameters, $\boldsymbol{Z} = [\boldsymbol{Z}_1 \boldsymbol{Z}_2 \cdots \boldsymbol{Z}_K]$ is the partitioned design matrix of random effects, with $\boldsymbol{Z}_k$ $n \times q_k$, the corresponding matrix design for the random effects k, $\boldsymbol{b}_k$ $(\boldsymbol{b}^{\text{T}} = (\boldsymbol{b}_1^{\text{T}}, \ldots, \boldsymbol{b}_K^{\text{T}}))$, $\widetilde{\boldsymbol{b}}_{(t)}^{\text{T}} = \left(\widetilde{\boldsymbol{b}}_{1(t)}^{\text{T}}, \ldots, \widetilde{\boldsymbol{b}}_{K(t)}^{\text{T}}\right)$, and $\sigma_{k(t)}^2, k = 1, \ldots, K$, are the current values of the variance parameters. Finally, for this specific model, the maximization updates for the beta coefficients and variance components are the same as before, where the variance components are

$$\sigma_{k(t+1)}^2 = \frac{1}{q_k}\sigma_{k(t)}^2\left[q_k - \sigma_{k(t)}^2 \text{tr}\left(\boldsymbol{A}_k \boldsymbol{Z}_k^{\text{T}} \boldsymbol{V}_{(t)}^{-1} \boldsymbol{Z}_k\right) + \sigma_{k(t)}^{-2} \widetilde{\boldsymbol{b}}_{k(t)}^{\text{T}} \boldsymbol{A}_k^{-1} \widetilde{\boldsymbol{b}}_{k(t)}\right], k = 1, \ldots, K.$$

These are obtained by maximizing $Q(\boldsymbol{\beta}, \boldsymbol{\theta} | \boldsymbol{\beta}_{(t)}, \boldsymbol{\theta}_{(t)})$ (defined above) with respect to $\sigma_k^2, k = 1, \ldots, K$.

5.2.1.2 REML

An alternative to the ML estimation of the variance components of model (5.1) and to avoid the underestimation of the maximum likelihood method is the restricted maximum likelihood estimation method (REML) proposed by Patterson and Thompson (1971). Among the several ways to define this, one is discussed by Laird and Ware (1982), which under a Bayesian paradigm, consists of estimating the parameters of the variance components by maximizing the marginal posterior distribution of the variance components by assuming a "locally" uniform prior to the distribution for $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, that is, $f(\boldsymbol{\beta}, \boldsymbol{\theta}) \propto 1$ (Pinheiro and Bates 2000). The marginal posterior of the variance components is given in Appendix 3

$$f(\boldsymbol{\theta}|\boldsymbol{y}) \propto \int f(\boldsymbol{\beta},\boldsymbol{\theta}|\boldsymbol{y})d\boldsymbol{\beta} \propto \int f(\boldsymbol{y}|\boldsymbol{\beta},\boldsymbol{\theta})d\boldsymbol{\beta}$$
$$\propto |\boldsymbol{V}|^{-\frac{1}{2}}\left|\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right|^{\frac{1}{2}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{y}-\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{V}^{-1}\left(\boldsymbol{y}-\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)\right\},$$

where $\widetilde{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}$, which corresponds to the maximum likelihood of the fixed effects when the variance components are assumed known as generalized least squares estimates of $\boldsymbol{\beta}$ (GLS). Then, the restricted maximum likelihood estimators (REML) $\boldsymbol{\theta}$ are those that maximize $f(\boldsymbol{\theta}|\boldsymbol{y})$, or equivalently, the REML $\boldsymbol{\theta}$ can be defined as maximizing the

$$\ell_R(\boldsymbol{\theta};\boldsymbol{y}) = -\frac{1}{2}\log\left(\left|\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right|\right) - \frac{1}{2}\log\left(|\boldsymbol{V}|\right) - \frac{1}{2}\left(\boldsymbol{y}-\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{V}^{-1}\left(\boldsymbol{y}-\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)$$

This function is known as the restricted likelihood because it can be shown that this also corresponds to the likelihood associated with the maximum number $(n-p-1)$ of linearly independent error contrasts $\boldsymbol{FY}$, where $\boldsymbol{F}$ is a full row rank $(n-p-1)\times n$ known matrix such that $\boldsymbol{FX} = \boldsymbol{0}$. It is important to point out that the associated likelihood based on the transformed data, $\boldsymbol{FY}$, gives the same result for any chosen contrast error matrix $\boldsymbol{F}$ and, consequently, this is invariant to fixed effect parameters (Harville 1974). Equivalently, the REML of $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ can be defined as those that maximize the

$$\ell_R(\boldsymbol{\beta},\boldsymbol{\theta};\boldsymbol{y}) = -\frac{1}{2}\log\left(\left|\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right|\right) - \frac{1}{2}\log\left(|\boldsymbol{V}|\right) - \frac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}\boldsymbol{V}^{-1}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})$$

This objective function is like the natural logarithm of likelihood function given in Eq. (5.2) (log-likelihood) except for the first term. To obtain the REML solutions or the maximum a posteriory (when adopting a locally uniform prior for the parameter, as described before) of the variance components parameters, (like for the MLE) numerical methods are required. See Jennrich and Schluchter (1986) and Lindstrom and Bates (1988) for details on the Newton–Raphson and Fisher Scoring algorithms; consult the lme4 R package (Bates et al. 2015) which uses a generic nonlinear optimizer and implements a large variety of different models (MLE and REML) that arise from the LMM when different structures of the variance of random effects and errors are adopted. For a derivation of the EM algorithm to obtain the REML, see Laird and Ware (1982) under model (5.1) for longitudinal data; this same approach can be used for the genomic model previously described, where $\boldsymbol{D} = \mathrm{Diag}\left(\sigma_1^2\boldsymbol{A}_1, \ldots, \sigma_K^2\boldsymbol{A}_K\right)$ and $\boldsymbol{R} = \sigma^2\boldsymbol{I}_n$. Consult Searle (1993) and Covarrubias-Pazaran (2016, 2018) for an implementation of this algorithm with the EM function in the sommer R package.

5.2.1.3 BLUPs

In many situations, in addition to the estimation of the fixed effects, the prediction of the random effects is also of interest. A standard method for "estimating" the random effects is the best linear unbiased predictor (BLUP; Robinson 1991), which originally was developed by Henderson (1975) in animal breeding for estimating merit in dairy cattle and is now commonly employed in many research areas (Piepho et al. 2008). If the variance components $\boldsymbol{D}$ and $\boldsymbol{R}$ are known, the best linear unbiased predictor (BLUP) of the random effects $\boldsymbol{b}$ is given by

$$\widetilde{\boldsymbol{b}}^{*} = \boldsymbol{D}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{V}^{-1}\left(\boldsymbol{y} - \boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right),$$

where $\widetilde{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}$ is the generalized least squared (GLS) estimator of $\boldsymbol{\beta}$. This can be obtained by maximizing with respect to $\boldsymbol{\beta}$ and $\boldsymbol{b}$, the joint density of $\boldsymbol{y}$ and $\boldsymbol{b}$, $f_{\boldsymbol{Y},\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{b})$, and is the reason why Harville (1985) called these estimates of realized values $\boldsymbol{b}$ (McLean et al. 1991), or likewise by solving the mixed model equations (MME) (Henderson 1975):

$$\begin{bmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{Z} \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{X} & \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{Z} + \boldsymbol{D}^{-1} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{\beta}} \\ \widetilde{\boldsymbol{b}}^{*} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y} \end{bmatrix}$$

from which the inversion of the variance–covariance matrix of $\boldsymbol{Y}$ is avoided, which can be helpful in some situations to save considerable computational resources. Note that $\widetilde{\boldsymbol{b}}^{*}$ corresponds to the posterior mean of the random effects where the fixed effects are replaced by its GLS (Searle et al. 2006).

When the variance components are unknown, which is most often the case, they are frequently estimated using restricted maximum likelihood estimators, which replace them in the corresponding equations. Then, the approximate best linear unbiased predictor is obtained and is referred to as the estimated or empirical best linear unbiased predictor (EBLUP) (Rencher 2008).

To solve the mixed model equations, there are several software packages that can be useful, but one in particular in the genomic context is the sommer package (Covarrubias-Pazaran 2016, 2018) that internally solves the MME after the variance components are estimated. The github version of the sommer R package can be accessed at https://github.com/cran/sommer and can be installed with the following commands:

```
install.packages('devtools');
library(devtools);
install_github('covaruber/sommer')
```

5.3 Linear Mixed Models in Genomic Prediction

In a simple genomic prediction context where $\boldsymbol{b}$ includes the genotype effects, and the genomic relationship matrix (VanRaden 2008), that is, $\boldsymbol{G}$ is available, very often the assumed variance–covariance matrix of the random effects is $\sigma_g^2\boldsymbol{G}$ and the errors are assumed as independently and identically distributed, $\boldsymbol{R} = \sigma^2\boldsymbol{I}_n$, where n is the total number of observations. In this case, the resultant model is known as the GBLUP model and when the pedigree is used, it is referred to as PBLUP. Another kind of information between lines can also be used, such as the relationship matrices derived from hyperspectral reflectance information (Krause et al. 2019). Other extensions of this model can be developed by taking into account other factors, for example, genotype× environment interaction, as will be illustrated later in the genomic prediction context.

In this case, where only the genotypic effects are taken into account, in the linear mixed model (5.1), the fixed effects design matrix is $\boldsymbol{X} = \boldsymbol{1}_n$, where the vector of length n corresponds to the general mean $\boldsymbol{\beta} = \beta_0$, $\boldsymbol{b} = (b_1, b_2, \ldots, b_J)^{\mathrm{T}}$ contains the genotypic effects of J lines, and $\boldsymbol{Z}$ is the incidence matrix design for the random line effects ($\boldsymbol{Z}_L$):

$$\boldsymbol{Y} = \boldsymbol{1}_n\mu + \boldsymbol{Z}_L\boldsymbol{b} + \boldsymbol{\epsilon}, \tag{5.3}$$

where $\boldsymbol{b} \sim N_J\left(\boldsymbol{0}, \sigma_g^2\boldsymbol{G}\right)$ and $\boldsymbol{R} = \sigma^2\boldsymbol{I}_n$.

The basic code to implement the GBLUP model (5.3) with the sommer package is the following:

```
A = mmer(y ~ 1,random= ~ vs(GID,Gu=G) , rcov= ~ vs(units),
         data=dat_F, verbose=FALSE)
```

where **y** and **GID** are the column names that contain the response variable and genotypes in data set **dat_F**. **G** is the genomic relationship matrix for lines which is specified in the Gu argument, that in general serves to provide a known variance–covariance matrix between the levels of the random effects (GID). In the "rcov" option, the argument "units" is always used to specify the error term.

5.4 Illustrative Examples of the Univariate LMM

Example 1 To illustrate the performance of the LMM in a genomic prediction context doing the fitting process with the sommer package, we considered a wheat data set that consisted of 500 markers measured for each line as the genomic information, and with 229 observations in total that registered grain yield (tons/ha): 30 lines in four environments with one or two repetitions.

Table 5.1 Prediction performance of the GBLUP model (5.3, M1) and the model (5.3) that results from ignoring the genomic information (M10): mean squared error of prediction (MSE) and Pearson's correlation (PC), and its standard deviation for each criterion is reported for each partition

	M1		M10	
PT	MSE	PC	MSE	PC
1	0.7025	0.6917	0.6924	0.6969
2	0.4583	0.6681	0.4407	0.7044
3	0.5075	0.5251	0.4923	0.5506
4	0.4719	0.6024	0.4468	0.6328
5	0.6433	0.5479	0.6433	0.5307
6	0.3657	0.5088	0.3569	0.5255
7	0.6923	0.5081	0.6777	0.5243
8	0.3285	0.5054	0.2952	0.57
9	0.4364	0.6855	0.429	0.6962
10	0.6807	0.6374	0.6513	0.6811
Average (SD)	0.5287 (0.14)	0.5881 (0.078)	0.5126 (0.143)	0.6112 (0.078)

The prediction performance of the model given in Eq. (5.3) (**M1**) was evaluated with 10 random partitions, where each partition was made up of two subsets, one containing 80% of the data and used for training the model, and the other containing the remaining 20% of the data and used to evaluate the prediction performance of the model in terms of the mean squared error of prediction (MSE).

Furthermore, this model assumes that the errors were independently and identically distributed as $\boldsymbol{\epsilon} \sim N_n(\mathbf{0}, \sigma^2 \boldsymbol{I}_n)$; independently of the genotypic effects, $\boldsymbol{b}$ was assumed multivariate normal with a null mean vector and a variance–covariance matrix equal to $\sigma_g^2 \boldsymbol{G}$, where the genomic relationship matrix was computed with the information of the 500 markers.

The variance components parameters, in this case σ_g^2 and σ^2, were estimated by restricted maximum likelihood estimation with the mmer function in the sommer package, using the default algorithm optimization, the Newton–Raphson method. For univariate response variables, the EM algorithm through the EM function in this R package can also be used.

The results are shown in Table 5.1, where we also present the Pearson's correlation (PC) and MSE of the same model but without taking into account the information of the genomic relationship between lines ($\boldsymbol{G}$), that is, the variance–covariance matrix for the genotypic effects is assumed to be $\text{Var}(\boldsymbol{b}) = \sigma_g^2 \boldsymbol{I}_J$. This model is referred to as **M10**. From this table, we can observe that model **M10** shows a slightly better performance in terms of both MSE and PC criteria than the **M1** model: the MSE of model **M1** is 3.15% greater than the MSE of **M10**, while the PC of the **M10** is 3.94% greater than the corresponding **M1** model. The better average performance was observed with model **M10**, which did not consider genomic information, suggests that the marker information in this particular case did not provide useful information; however, in general, this is not expected when using

marker information for prediction, although this could change with larger data sets (more lines and more markers) or by improving the quality of the available data.

The R code to reproduce this result is given in Appendix 4. This can be adapted easily to another CV strategy of interest where the objective, for example, can be the prediction of non-observed lines in some environments or the prediction of lines in a future year.

An extension of the GBLUP model is the G×E BLUP model that takes into account the main environmental effects, the genotypic effects, and the genotype ×environment interaction effects:

$$Y = \mathbf{1}_n\mu + \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{Z}_L\boldsymbol{b}_1 + \boldsymbol{Z}_{EL}\boldsymbol{b}_2 + \boldsymbol{\epsilon} \tag{5.4}$$

where now the fixed effects are part of the linear mixed model (5.1) that was explicitly split into the general mean part ($\mathbf{1}_n\mu$) and the environment effects term ($\boldsymbol{X}_E\boldsymbol{\beta}_E$), $\boldsymbol{X} = [\mathbf{1}_n\, \boldsymbol{X}_E]$ and $\boldsymbol{\beta} = \left(\mu, \boldsymbol{\beta}_E^{\mathrm{T}}\right)^{\mathrm{T}}$. Similarly, for the random effects, $\boldsymbol{Z} = [\boldsymbol{Z}_L\, \boldsymbol{Z}_{EL}]$ and $\boldsymbol{b} = \left[\boldsymbol{b}_1^{\mathrm{T}}, \boldsymbol{b}_2^{\mathrm{T}}\right]^{\mathrm{T}}$, where $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ were the vectors with the random genotypic effects and the vector with the random genotype×environment interaction effects, with incidence matrix $\boldsymbol{Z}_L$ and $\boldsymbol{Z}_{EL}$, respectively. For $\boldsymbol{b}_1$, the same distribution as the GBLUP model was assumed, $\boldsymbol{b}_1 \sim N_J\left(\mathbf{0}, \sigma_g^2\boldsymbol{G}\right)$, and for the second random effect, $\boldsymbol{b}_2 \sim N_J(\mathbf{0}, \boldsymbol{\Sigma}_E \otimes \boldsymbol{G})$, where $\boldsymbol{\Sigma}_E \otimes \boldsymbol{G}$ is the relationship matrix of the genotype×environment interaction term, with $\boldsymbol{\Sigma}_E$ the genetic variance–covariance matrix between I environments; the ith element of the diagonal of $\boldsymbol{\Sigma}_E$, σ_{Ei}^2, is the genetic variance in environment i, $i = 1, \ldots, I$, and $\sigma_{Eik}\boldsymbol{G}$ is the genetic variance–covariance matrix for lines in environments i and k, where σ_{Eik} is the element (i, k) of $\boldsymbol{\Sigma}_E$.

When $\boldsymbol{\Sigma}_E$ has a non-diagonal structure, the information from the genomic relationship matrix and the correlated environments can be helpful for improving the prediction performance of the model by borrowing information between lines inside an environment and between lines across and among environments (Burgueño et al. 2012).

Example 2 To illustrate how model (5.4) can be implemented using the sommer package, the same data used in Example 1 are considered, where the same 30 genotypes are in the four environments. Besides the line indicator (GID), environment information (Env) was also available in the data set, which was needed for implementing model (5.4). The adopted structure for the variance–covariance matrix between environments is $\boldsymbol{\Sigma}_E = \sigma_{EG}^2\boldsymbol{I}_I$ and the resulting model is referred to as M2. Another explored model (M20) was obtained under the same specification, with the difference that $\boldsymbol{G}$ was set equal to the identity matrix.

Using the same validation scheme that was used in Example 1, the results for each of the 10 random partitions are shown in Table 5.2, in which, for illustrative purposes, model (5.3) plus environment as a fixed effect (M11) is also included, that is,

Table 5.2 Prediction performance of two sub-models of (5.4): model M2 in which $\text{Var}(\boldsymbol{b}_1) = \sigma_g^2 \boldsymbol{G}$, $\boldsymbol{b}_2 \sim N_J(\boldsymbol{0}, \boldsymbol{\Sigma}_E \otimes \boldsymbol{G})$ and $\boldsymbol{\Sigma}_E = \sigma_{EG}^2 \boldsymbol{I}_I$ (M2); and model M20 that is the same as model M2 but the genomic information is not taken into account, that is, $\boldsymbol{G} = \boldsymbol{I}_J$

	M2		M20		M11	
PT	MSE	PC	MSE	PC	MSE	PC
1	0.5641	0.8198	0.5364	0.8576	0.5658	0.8168
2	0.4292	0.7645	0.4014	0.8393	0.4447	0.6424
3	0.3865	0.7873	0.3621	0.8497	0.4246	0.6422
4	0.45	0.6997	0.401	0.7897	0.3853	0.6992
5	0.6879	0.5307	0.6636	0.5902	0.595	0.5933
6	0.3048	0.6802	0.2819	0.7326	0.3267	0.5777
7	0.6405	0.6823	0.6369	0.7066	0.558	0.6914
8	0.4132	0.5261	0.4053	0.5562	0.2999	0.5817
9	0.3217	0.8384	0.2978	0.8869	0.3457	0.7797
10	0.5397	0.8368	0.5206	0.8572	0.5647	0.7307
Average (SD)	0.4738 (0.13)	0.7166 (0.116)	0.4507 (0.133)	0.7666 (0.117)	0.451 (0.112)	0.6755 (0.083)

M11 is referred to as model (5.3) plus environment effects (Env). The mean squared error of prediction (MSE) and Pearson's correlation (PC) for each partition are reported. SD is the standard deviation

$$\boldsymbol{Y} = \boldsymbol{1}_n \mu + \boldsymbol{X}_E \boldsymbol{\beta}_E + \boldsymbol{Z}_L \boldsymbol{b} + \boldsymbol{\epsilon},$$

where μ, $\boldsymbol{\beta}_E$, and $\boldsymbol{b}$ are as before (5.3), and $\boldsymbol{X}_E \boldsymbol{\beta}_E$ is the predictor term corresponding to the environment fixed effects.

From Table 5.2 we can observe yet again a moderately better performance of model M20 that does not take into account the genomic information. Model M2 also confirms the lack of usefulness of the marker information in this case, but again, this in general is expected to change for other data sets with a greater number of lines, markers, or more data quality. The MSE of model M2 is 5.11% greater than the MSE of model M20, while the PC value of this last model is 6.98% greater than the corresponding PC value obtained with the M20 model. When comparing the M20 model and M11, the MSE of this last model is just 0.075% greater than the corresponding MSE of M20, but when considering the PC value, the first model resulted in 13.98% greater than model (5.1) plus the environment effect. Indeed, because of the high variation observed across partitions (SD values of PC and MSE), there is no significant difference between models in Table 5.2.

Furthermore, in terms of the average MSE, the model with the best performance between those presented in Table 5.1 (M10) is 13.73% greater than the average MSE of the best performance model between those compared in Table 5.2 (M11), while in terms of the average Pearson's correlation, the best model in Table 5.2 (M20) is 25.42% greater than the average Pearson's correlation of the best model in Table 5.1. The worse average MSE performance of those in Table 5.1 (M1) is 17.31% greater than the best average MSE performance in Table 5.2 (M20), and the best average PC

performance in Table 5.2 (M20) is 30.37% greater than the worse average PC performance in Table 5.1 (M1). Actually, the best average MSE model in Table 5.1 (M10) is 8.19 greater than the worse average MSE model in Table 5.2 (M2), while the worse average PC model in Table 5.2 (M11) is 10.51% greater than the average PC model in Table 5.1 (M10).

The R code to reproduce these results is given in Appendix 5.

Other versions of model (5.4) can be obtained by adopting other variance–covariance structures. For example, another version of model (5.4) can be obtained when environment covariates are available ($\boldsymbol{W}$) and they are used to model the $G\times E$ predictor term, specifically when the genetic variance–covariance matrix between environments, $\boldsymbol{\Sigma}_E$, is modeled by $\boldsymbol{\Sigma}_E = \sigma^2_{EG}\boldsymbol{O}$, where $\boldsymbol{O} = \frac{1}{p_w}\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}$ and the similarity between environments is computed like the genomic relationship matrix (**G**), using the information of p_w environment covariates (Jarquín et al. 2014; Martini et al. 2020), or where $\boldsymbol{O}$ is obtained from phenotypic correlations across environments from related historical data (Martini et al. 2020). In the sommer package, this can be implemented using the following basic R code:

```
O = diag(I) #Specified the O matrix for the I environments
dat_F$Env_GID = paste(dat_F$Env,dat_F$GID,sep='_')
GE = kronecker(O,G)
rnGWE = expand.grid(row.names(G),unique(dat_F$Env))
row.names(GE) = paste(rnGWE[,2],rnGWE[,1],sep='_')
colnames(GE) = row.names(GE)
A = mmer(y ~ Env, random= ~ vs(GID,Gu=G) + vs(Env_GID,Gu = GE),
      rcov= ~ vs(units), data=dat_F)
```

Other more complex models can be explored with the sommer package when more data information is available, such as specifying an unstructured variance–covariance matrix for $\boldsymbol{\Sigma}_E$. A simpler model is the non-correlated heterogeneous variance components (for environments) which arises by assuming a diagonal structure, $\boldsymbol{\Sigma}_E = \mathrm{Diag}\left(\sigma^2_1, \ldots, \sigma^2_I\right)$. This can be implemented by replacing the interaction term in the predictor in sommer *vs(**Env**,Gu = **G**)* by *vs(ds(**Env**), **GID**,Gu = **G**)*. Similarly, for a specific environment residual variance, *vs(units)* need to be replaced by *vs(ds(Env),units)* or *vs(at(Env),units)*. See Appendix 7 for a basic code to implement all these models and see Covarrubias-Pazaran (2018) for more variance structures that can be exploited in this model.

5.5 Multi-trait Genomic Linear Mixed-Effects Models

In some genomic applications, there are several traits of interest and all of them are measured in some lines but in other lines only subsets of those traits are measured. Although separate univariate genomic linear mixed models can be performed to analyze all measured traits, sometimes single univariate genomic models do not

work well, especially in traits with low heritability. When low heritability traits have at least moderate correlation with high heritability traits, the prediction performance ability for these low heritability traits could strongly increase by using a multi-trait model (Jia and Jannink 2012; Montesinos-López et al. 2016; Budhlakoti et al. 2019).

If for each line ($j = 1, \ldots J$), n_T traits are measured, $Y_{jt}, t = 1, \ldots n_T$, the multi-trait genomic linear mixed-effects model adopts an unstructured covariance matrix for the residuals between traits and for the random genotypic effects between traits, and similar to the univariate trait models (5.3), this can be expressed as

$$\begin{bmatrix} Y_{j1} \\ Y_{j2} \\ \vdots \\ Y_{jn_T} \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_{n_T} \end{bmatrix} + \begin{bmatrix} g_{j1} \\ g_{j2} \\ \vdots \\ g_{jn_T} \end{bmatrix} + \begin{bmatrix} \epsilon_{j1} \\ \epsilon_{j2} \\ \vdots \\ \epsilon_{jn_T} \end{bmatrix}, j = 1, \ldots, J, \tag{5.5}$$

where μ_t, $t = 1, \ldots, n_T$, are the specific trait means, g_{jt}, $t = 1, \ldots, n_T$, are the specific trait genotypic effects, and ϵ_{jt}, $t = 1, \ldots, n_T$, are the random error terms corresponding to each trait. Furthermore, $\boldsymbol{b} = \left[\boldsymbol{g}_1^{\mathrm{T}}, \ldots, \boldsymbol{g}_J^{\mathrm{T}}\right]^{\mathrm{T}} \sim N(\mathbf{0}, \boldsymbol{G} \otimes \boldsymbol{\Sigma}_{\mathrm{T}})$, $\boldsymbol{g}_j = \left[g_{j1}, \ldots, g_{jn_T}\right]^{\mathrm{T}}$, $j = 1, \ldots, J$, and $\boldsymbol{\epsilon}_j = \left[\epsilon_{j1}, \ldots, \epsilon_{jn_T}\right]^{\mathrm{T}}$, $j = 1, \ldots, J$, are independent multivariate normal random vectors with null mean and variance $\mathbf{R}_{n_T}$, $\boldsymbol{\Sigma}_T$ is $n_T \times n_T$ matrix that represents the genetic covariance between traits, and $\otimes$ is the Kronecker product.

In matrix notation, it is the linear mixed model (5.1) where $\boldsymbol{Y} = \left[\boldsymbol{Y}_1^{\mathrm{T}}, \ldots, \boldsymbol{Y}_J^{\mathrm{T}}\right]^{\mathrm{T}}$, $\boldsymbol{Y}_j = \left[Y_{j1}, \ldots, Y_{jn_T}\right]^{\mathrm{T}}$, $\boldsymbol{X} = \mathbf{1}_J \otimes \boldsymbol{I}_{n_T}$, $\boldsymbol{\beta} = \boldsymbol{\mu} = \left(\mu_1, \ldots, \mu_{n_T}\right)^{\mathrm{T}}$, $\boldsymbol{Z} = \boldsymbol{I}_{n_T J}$, $\boldsymbol{b} = \left[\boldsymbol{g}_1^{\mathrm{T}}, \ldots, \boldsymbol{g}_J^{\mathrm{T}}\right]^{\mathrm{T}}, \boldsymbol{\epsilon} = \left[\boldsymbol{\epsilon}_1^{\mathrm{T}} \ldots \boldsymbol{\epsilon}_J^{\mathrm{T}}\right]^{\mathrm{T}} \sim N(\mathbf{0}, \boldsymbol{I}_J \otimes \boldsymbol{R}_{n_T})$, and $\boldsymbol{b} = \left[\boldsymbol{g}_1^{\mathrm{T}}, \ldots, \boldsymbol{g}_J^{\mathrm{T}}\right]^{\mathrm{T}} \sim N(\mathbf{0}, \boldsymbol{G} \otimes \boldsymbol{\Sigma}_{\mathrm{T}})$. Similarly, the extended model that arises by adding more fixed effects ($\boldsymbol{X}$) can be specified by adding a term $\boldsymbol{X\beta}$ to the predictor:

$$\boldsymbol{Y} = (\mathbf{1}_{IJ} \otimes \boldsymbol{I}_{n_T})\boldsymbol{\mu} + \boldsymbol{X\beta} + \boldsymbol{Zb} + \boldsymbol{\epsilon} \tag{5.5a}$$

When $\boldsymbol{\Sigma}_T$ and $\boldsymbol{R}$ are diagonal matrices, model (5.5) is equivalent to separately fitting a univariate GBLUP model to each trait.

The R code to fit this multivariate model with the sommer package is

```
A = mmer(cbind(T1,...,TnT) ~x1+x2+...+xp ,
         random= ~ vs(GID,Gu=G), rcov= ~ vs(units), data=dat_F)
```

where **y** and **GID** are again the column names corresponding to the response variables and genotypes in data set **dat_F**, while *T1*, …, *TnT* are the column names of the matrix of response variables (**y**) in **dat_F** corresponding to the traits to be used, and similarly, *x1*, *x2*,…, *xp* are the column names of *p* covariates to be included in the fitting process (see below the R code for Example 3). The rest of the arguments are the same as those described in the R code of model 5.3.

Example 3 To illustrate the fitting of the multi-trait genomic model (5.5) (M3), we considered the same data set used in Examples 1 and 2, but with the addition of trait (y2) to be able to explore the implementation of a bivariate trait genomic model. The same CV strategy implemented in Example 1 was used. In addition to model (5.5), we also evaluated a sub-model that was obtained by considering a diagonal structure for $\boldsymbol{\Sigma}_T$, that is, $\boldsymbol{\Sigma}_T = \text{Diag}\left(\sigma^2_{T_1}, \sigma^2_{T_2}\right)$. This model will be referred to as M32.

The results are in Table 5.3. On average, the two evaluated models (M3 and M32) showed a similar performance in terms of the two criteria used, MSE and PC, for both traits, but in all partitions a slightly better performance was observed in favor of model M3. For trait T1, the simpler model (M32) gave an MSE 0.785% greater than model M3, while the more complex model (M3) gave a PC only 1.066% greater than that of model M32. The difference was less for the second trait (T2), where the average MSE of M32 was only 0.165% greater than the one corresponding to model M3, while the PC of M3 was only 0.046% greater than the PC of M32.

Furthermore, note that the difference between the univariate models presented in Tables 5.1 and 5.2 and the multivariate models of Table 5.3 is not significant (only on average the models in Table 5.2 result better than models in Table 5.1) because the large standard deviation observed across partitions in MSE and PC, which in this case indicate that the multivariate model does not help improve the prediction accuracy in the trait of interest (first trait). But as commented before, this benefit could be obtained with more related auxiliary secondary traits and larger data sets of good quality.

Table 5.3 Prediction performance of a bivariate trait model (5.5)

Models	M3				M32			
Trait	T1		T2		T1		T2	
PT	MSE	PC	MSE	PC	MSE	PC	MSE	PC
1	0.6959	0.6954	0.0978	0.984	0.6973	0.6939	0.0982	0.9835
2	0.4476	0.6851	0.1411	0.9256	0.4526	0.6759	0.1409	0.9253
3	0.494	0.5448	0.0446	0.9846	0.503	0.5315	0.0436	0.985
4	0.4636	0.6125	0.1258	0.9665	0.4674	0.6079	0.1266	0.9663
5	0.6398	0.5496	0.1588	0.9306	0.6413	0.5489	0.1596	0.93
6	0.3598	0.5192	0.0691	0.9775	0.3632	0.5134	0.0693	0.9772
7	0.6864	0.514	0.0273	0.9953	0.691	0.5094	0.0272	0.9951
8	0.3104	0.5421	0.2011	0.9492	0.3187	0.5267	0.2	0.9482
9	0.4394	0.678	0.1983	0.9468	0.4359	0.6833	0.2005	0.9452
10	0.6685	0.6575	0.0838	0.974	0.6759	0.644	0.0837	0.9738
Average (SD)	0.5205 (0.142)	0.5998 (0.074)	0.1148 (0.061)	0.9634 (0.024)	0.5246 (0.141)	0.5935 (0.076)	0.115 (0.061)	0.963 (0.024)

This model is referred to as M3 and when assuming a diagonal structure for $\boldsymbol{\Sigma}_T$, $\boldsymbol{\Sigma}_T = \text{Diag}\left(\sigma^2_{1T}, \sigma^2_{2T}\right)$, it is referred to as M32. The mean squared error of prediction (MSE) and Pearson's correlation (PC) for each trait in each partition are reported. SD is the standard deviation

The R code to obtain the results given in Table 5.3 is provided in Appendix 6. At the end of this Appendix, in the comment lines, the code is also available for a CV strategy, when we are interested in evaluating the performance of a bivariate model where only trait y2 is missing in testing data set and all the information of the other trait (y1) is available. This could be useful in real applications where the interest lies in predicting traits that are difficult or expensive to measure, with the phenotypic information of correlated traits that are easy or inexpensive to measure (Calus and Veerkamp 2011; Jiang et al. 2015). Of course, the code could be adapted for any other relevant strategy.

In a similar fashion, just as univariate genomic linear mixed model (5.4), model (5.5) can be directly extended to a model that considers the genotype ×environment interaction term. Next, we do this for the balanced case, and for this we assume that for each environment $i = 1, \ldots, I$, J lines were phenotyped for n_T traits, Y_{ijt}, $t = 1, \ldots, n_T$. In matrix notation, the extended $G\times E$ model (5.4) plus fixed effects ($\boldsymbol{X\beta}$) is given by

$$\boldsymbol{Y} = (\mathbf{1}_{IJ} \otimes \boldsymbol{I}_{\mathrm{n_T}})\boldsymbol{\mu} + \boldsymbol{X\beta} + \boldsymbol{Z}_L\boldsymbol{b}_1 + \boldsymbol{Z}_{EL}\boldsymbol{b}_2 + \boldsymbol{\epsilon}, \tag{5.6}$$

where $\boldsymbol{Y} = \left[\boldsymbol{Y}_1^{\mathrm{T}} \ldots \boldsymbol{Y}_I^{\mathrm{T}}\right]^{\mathrm{T}}$, $\boldsymbol{Y}_i = \left[\boldsymbol{Y}_{i1}^{\mathrm{T}}, \ldots, \boldsymbol{Y}_{iJ}^{\mathrm{T}}\right]^{\mathrm{T}}$, $\boldsymbol{Y}_{ij} = \left[\boldsymbol{Y}_{ij1}, \ldots \boldsymbol{Y}_{ijn_T}\right]^{\mathrm{T}}$, $i = 1, \ldots, I$, $j = 1, \ldots, J$, $\mathbf{1}_{IJ}$ is the vector of ones of order IJ, $\boldsymbol{I}_{n_T}$ is the identity matrix of dimension n_T, $\boldsymbol{\mu} = \left(\mu_1, \ldots, \mu_{n_T}\right)^{\mathrm{T}}$ is the vector with the general specific trait means, $\boldsymbol{Z}_L = \mathbf{1}_I \otimes \boldsymbol{I}_{\mathrm{n_T}}$ and $\boldsymbol{Z}_{EL} = \boldsymbol{I}_{IJn_T}$ are the incidence matrices of genotype random effects ($\boldsymbol{b}_1$) and the incidence matrices of the genotype×environment interactions random effects ($\boldsymbol{b}_2$), respectively, with $\boldsymbol{b}_1 = \left[\boldsymbol{g}_1^{\mathrm{T}}, \ldots, \boldsymbol{g}_J^{\mathrm{T}}\right]^{\mathrm{T}}$ and $\boldsymbol{b_2} = \left[\boldsymbol{g}_{21}^{\mathrm{T}}, \ldots, \boldsymbol{g}_{2I}^{\mathrm{T}}\right]^{\mathrm{T}}$, $\boldsymbol{g}_j = \left[g_{j1}, \ldots, g_{jn_T}\right]^{\mathrm{T}}$, $\boldsymbol{g}_{2i} = \left[\boldsymbol{g}_{2i1}^{\mathrm{T}}, \ldots, \boldsymbol{g}_{2iJ}^{\mathrm{T}}\right]^{\mathrm{T}}$, and $\boldsymbol{g}_{2ij} = \left[\boldsymbol{g}_{2ij1}, \ldots \boldsymbol{g}_{2ijn_T}\right]^{\mathrm{T}}$, $i = 1, \ldots, I$, $j = 1, \ldots, J$. In addition, it is assumed that $\boldsymbol{\epsilon} = \left[\boldsymbol{\epsilon}_1^{\mathrm{T}} \ldots \boldsymbol{\epsilon}_I^{\mathrm{T}}\right]^{\mathrm{T}} \sim N(\mathbf{0}, \boldsymbol{I}_{IJ} \otimes \boldsymbol{R}_{\mathrm{n_T}})$, $\boldsymbol{b}_1 \sim N(\mathbf{0}, \boldsymbol{G} \otimes \boldsymbol{\Sigma}_T)$, and $\boldsymbol{b}_2 \sim N(\mathbf{0}, \boldsymbol{\Sigma}_E \otimes \boldsymbol{G} \otimes \boldsymbol{\Sigma}_{2T})$.

This shows that when $\boldsymbol{\Sigma}_T$, $\boldsymbol{\Sigma}_{2T}$, $\boldsymbol{\Sigma}_E$, and $\boldsymbol{R}$ are diagonal matrices, model (5.6) is equivalent to separately fitting a univariate GBLUP model for each trait.

Example 4 To illustrate the fitting and evaluation process of model (5.6), we considered a data set that contains the information of two traits, for which 150 lines were phenotyped each in two environments, and given a total of 300 bivariate phenotypic data points. Also, a genomic relationship matrix for the lines is available that was computed with marker information.

The first explored model is referred to as M4 and assumes an unstructured variance–covariance matrix for all the components in model (5.6), except for the assumption that $\boldsymbol{\Sigma}_E = \boldsymbol{I}_I$, i.e., the model assumes the same variance–covariance among environments. In addition to this model (M4), three sub-models were also explored: M42, which considers a diagonal structure for the genetic variance–covariance between traits, $\boldsymbol{\Sigma}_{2T} = \mathrm{Diag}\left(\sigma^2_{1T_1}, \sigma^2_{1T_2}\right)$, model M43 in which $\boldsymbol{\Sigma}_{1T} = \mathrm{Diag}\left(\sigma^2_{1T_1}, \sigma^2_{1T_2}\right)$ and $\boldsymbol{\Sigma}_{2T} = \mathrm{Diag}\left(\sigma^2_{2T_1}, \sigma^2_{2T_2}\right)$, and model M44 which is the same as

M43 but with an assumed diagonal variance–covariance matrix structure for the error, $\boldsymbol{R} = \text{Diag}\left(\sigma^2_{e_1}, \sigma^2_{e_2}\right)$.

The results are shown in Table 5.4, from which we can observe that for trait 1 (GY), the best performance under both criteria (MSE and PC) was obtained with the more complex model: M4. For this trait (GY), the MSE of models M42, M43, and M44 were 7.53%, 8.21%, and 8.17%, respectively, greater than the MSE of

Table 5.4 Prediction performance of some sub-models of model (5.6)

Model	M4				M42			
Trait	T1 (GY)		T2 (Testwt)		T1 (GY)		T2 (Testwt)	
PT	MSE	PC	MSE	PC	MSE	PC	MSE	PC
1	0.1895	0.5962	1.0291	0.6416	0.1717	0.5691	0.9357	0.6942
2	0.2414	0.5091	0.8139	0.6466	0.2444	0.5127	0.7199	0.6965
3	0.2638	0.8089	1.2914	0.5875	0.2324	0.5661	1.0839	0.7705
4	0.475	0.1266	0.9975	0.7118	0.4856	0.101	1.0271	0.6846
5	0.2728	0.2237	0.8053	0.6897	0.272	0.2402	0.8	0.683
6	0.2919	0.7391	1.031	0.5288	0.29	0.373	0.6211	0.7809
7	0.2413	0.4145	1.0778	0.7836	0.3155	0.2617	1.1276	0.7808
8	0.1696	0.7954	1.0456	0.538	0.1785	0.4238	1.1182	0.726
9	0.2412	0.8813	1.0937	0.4014	0.3293	0.3998	1.9201	0.6492
10	0.2078	0.9006	1.4155	0.3746	0.2704	0.6108	1.7571	0.6505
Average (SD)	0.2594 (0.085)	0.5995 (0.275)	1.0601 (0.186)	0.5904 (0.132)	0.279 (0.089)	0.4058 (0.166)	1.1111 (0.422)	0.7116 (0.051)
Model	M43				M44			
Trait	T1 (GY)		T2 (Testwt)		T1 (GY)		T2 (Testwt)	
PT	MSE	PC	MSE	PC	MSE	PC	MSE	PC
1	0.1717	0.5602	0.9354	0.7015	0.1716	0.5438	0.9623	0.7053
2	0.25	0.4885	0.7258	0.6987	0.2515	0.4383	0.7536	0.6907
3	0.2344	0.5482	1.1496	0.7698	0.2324	0.5182	1.2674	0.7631
4	0.4956	0.0226	1.0861	0.6277	0.4852	−0.055	1.1752	0.5615
5	0.274	0.2154	0.806	0.6778	0.2738	0.1416	0.8357	0.66
6	0.2895	0.3758	0.6211	0.7803	0.2909	0.3468	0.6096	0.7864
7	0.2925	0.319	1.0604	0.7936	0.3001	0.2746	1.0997	0.7856
8	0.1782	0.4247	1.1177	0.7261	0.1828	0.3739	1.1289	0.7139
9	0.324	0.4071	1.9091	0.6488	0.3306	0.3438	1.9212	0.6499
10	0.2974	0.5764	1.8732	0.6506	0.2874	0.5483	1.9158	0.6434
Average (SD)	0.2807 (0.091)	0.3938 (0.173)	1.1284 (0.439)	0.7075 (0.059)	0.2806 (0.088)	0.3474 (0.191)	1.1669 (0.445)	0.696 (0.071)

Model (5.6) with $\boldsymbol{\Sigma}_E = \boldsymbol{I}_I$ is referred to as M4, and if additionally, $\boldsymbol{\Sigma}_{2T} = \text{Diag}\left(\sigma^2_{1T_1}, \sigma^2_{1T_2}\right)$, the model is referred to as M42. Model M42 but with $\boldsymbol{\Sigma}_{1T} = \text{Diag}\left(\sigma^2_{1T_1}, \sigma^2_{1T_2}\right)$ is referred to as M43, and model M44 is the same as M43 but with $\boldsymbol{R} = \text{Diag}\left(\sigma^2_{e_1}, \sigma^2_{e_2}\right)$. The mean squared error of prediction (MSE) and Pearson's correlation (PC) for each trait in each partition are reported. SD is the standard deviation

model M4. For the same trait (GY), the PC of model M4 was 47.73%, 52.24%, and 72.57%, greater than the PC of models M42, M43, and M44, respectively.

For the second trait (Testwt), model M4 also showed the best performance, but only under the MSE criteria: the MSE of models M42, M43, and M44 were 4.81%, 6.44%, and 10.08%, respectively, greater than the MSE corresponding to model M4, also suggesting an increasing degradation pattern in the MSE performance as the model became simpler with fewer parameters to estimate in relation to M4. In terms of PC, the best performance was achieved with model M42, which gave 20.54%, 0.583%, and 2.247% greater performances than models M4, M43, and M44, respectively.

Appendix 7 shows the R code used to reproduce the results in Table 5.4 with the sommer package. At the end of this code, we also included the basic code to explore other variance–covariance structures. Specifically, this is the code used to explore the model with heterogeneous genetic variance–covariance matrix, $\boldsymbol{\Sigma}_{2T}$, across environments, that is, $\boldsymbol{g}_{2i} \sim N(\mathbf{0}, \boldsymbol{G} \otimes \boldsymbol{\Sigma}_{2iT})$, $i = 1, \ldots, I$, which are assumed independent across environments.

5.6 Final Comments

The multi-trait linear model proposed by Henderson and Quaas (1976) in animal breeding can bring benefits in comparison to single-trait modeling for the improvement of prediction accuracy when incorporating correlated traits, as well as for obtaining an optimal and simplified total merit selection index (Okeke et al. 2017).

When the goal is to predict difficult or expensive traits that are correlated with inexpensive secondary traits, the use of multi-trait models could be helpful in developing better genomic selection strategies. Similarly, improvement of the accuracy of prediction for low-heritability key traits can follow from the use of high-heritability secondary traits (Jia and Jannink 2012; Muranty et al. 2015). Furthermore, this can be combined with the information of traits obtained using the speed breeding methodology to shorten the breeding cycles and accelerate breeding programs (Ghosh et al. 2018; Watson et al. 2019).

While the advantage of the multi-trait model is clearly documented, larger data sets and more computing resources are required, as there are additional parameters that need to be estimated (genetic and error covariances), which may affect the accuracy of genomic prediction. Additionally, convergence problems often arise when implementing complex mixed linear models and especially when small data sets are used.

Although the application of multi-trait models can be easily adapted with regard to genetic correlation, heritability, training population composition, and size of data sets, some other factors need to be carefully considered when using these methods for the improvement of genomic accuracy predictions (Lorenz and Smith 2015; Covarrubias-Pazaran et al. 2018). For example, biased and suboptimal choices between univariate and multi-trait models can result from using auxiliary traits that

are measured on individuals to be tested, but appropriate cross-validations strategies could be helpful in determining the usefulness of combining the multi-trait information with multi-trait models (Runcie and Cheng 2019).

Appendix 1

$$
\begin{aligned}
f_{\boldsymbol{b}|\boldsymbol{Y}}(\boldsymbol{b}|\boldsymbol{y}) &\propto f_{\boldsymbol{Y}|\boldsymbol{b}}(\boldsymbol{y}|\boldsymbol{b}) f_{\boldsymbol{b}}(\boldsymbol{b}) \\
&\propto \frac{1}{(2\pi\sigma^2)^{-\frac{n}{2}}} \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{X\beta}-\boldsymbol{Zb})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X\beta}-\boldsymbol{Zb}) - \frac{1}{2}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{b}\right] \\
&\propto \exp\left[-\frac{1}{2}\boldsymbol{b}^{\mathrm{T}}\left(\boldsymbol{D}^{-1}+\sigma^{-2}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\right)\boldsymbol{b} - \sigma^{-2}(\boldsymbol{y}-\boldsymbol{X\beta})^{\mathrm{T}}\boldsymbol{Zb}\right] \\
&\propto \exp\left[-\frac{1}{2}\left(\boldsymbol{b}-\widetilde{\boldsymbol{b}}\right)^{\mathrm{T}}\widetilde{\boldsymbol{D}}^{-1}\left(\boldsymbol{b}-\widetilde{\boldsymbol{b}}\right)\right]
\end{aligned}
$$

$\widetilde{\boldsymbol{D}} = \left(\boldsymbol{D}^{-1} + \sigma^{-2}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\right)^{-1}$ and $\widetilde{\boldsymbol{b}} = \sigma^{-2}\widetilde{\boldsymbol{D}}\boldsymbol{Z}^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X\beta})$. So $\boldsymbol{b} \mid \boldsymbol{Y}=\boldsymbol{y} \sim N_q\left(\widetilde{\boldsymbol{b}}, \widetilde{\boldsymbol{D}}\right)$.

Appendix 2

Because $(\boldsymbol{ABA}^{\mathrm{T}} + \boldsymbol{C})^{-1} = \boldsymbol{C}^{-1} - \boldsymbol{C}^{-1}\boldsymbol{A}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{A} + \boldsymbol{B}^{-1})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{C}^{-1}$ (Johnson and Wichern (2002)), then

$$
\widetilde{\boldsymbol{D}} = \left(\boldsymbol{D} + \sigma^{-2}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\right)^{-1} = \boldsymbol{D} - \boldsymbol{DZ}^{\mathrm{T}}\left(\boldsymbol{ZDZ}^{\mathrm{T}} + \sigma^2\boldsymbol{I}_n\right)^{-1}\boldsymbol{ZD}
$$

and thus

$$
\begin{aligned}
\mathrm{tr}\left(\widetilde{\boldsymbol{D}}^{(t)}\boldsymbol{D}^{-1}\right) &= \mathrm{tr}\left[\left(\boldsymbol{D}^{(t)} - \boldsymbol{D}^{(t)}\boldsymbol{Z}^{\mathrm{T}}\left(\boldsymbol{ZD}^{(t)}\boldsymbol{Z}^{\mathrm{T}} + \sigma^{2(t)}\boldsymbol{I}_n\right)^{-1}\boldsymbol{ZD}^{(t)}\right)\boldsymbol{D}^{-1}\right] \\
&= \sum_{k=1}^{K}\frac{\sigma_k^{2(t)}}{\sigma_k^2}\left[q_k - \sigma_k^{2(t)}\mathrm{tr}\left(\boldsymbol{A}_k\boldsymbol{Z}_k^T\boldsymbol{V}^{-(t)}\boldsymbol{Z}_k\right)\right],
\end{aligned}
$$

where $\boldsymbol{V}^{-(t)} = (\boldsymbol{ZD}^{(t)}\boldsymbol{Z}^{\mathrm{T}} + \sigma^{2(t)}\boldsymbol{I}_n)^{-1}$.

Appendix 3

Because

$$\exp\left[-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{X\beta})^{\mathrm{T}}\boldsymbol{V}^{-1}(\boldsymbol{y}-\boldsymbol{X\beta})\right]=\exp\left\{-\frac{1}{2}\left[\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X\beta}-2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X\beta}+\boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}\right]\right\}$$
$$=\exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)+\frac{1}{2}\widetilde{\boldsymbol{\beta}}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\widetilde{\boldsymbol{\beta}}-\frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}\right\},$$

where $\widetilde{\boldsymbol{\beta}}=\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}$,

$$\begin{aligned}
f(\boldsymbol{\theta}|\boldsymbol{y}) &\propto \int \frac{|\boldsymbol{V}|^{-\frac{1}{2}}}{(2\pi)^{\frac{q}{2}}}\exp\left[-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{X\beta})^{\mathrm{T}}\boldsymbol{V}^{-1}(\boldsymbol{y}-\boldsymbol{X\beta})\right]d\boldsymbol{\beta}\\
&\propto |\boldsymbol{V}|^{-\frac{1}{2}}\left|\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right|^{-1/2}\exp\left\{-\frac{1}{2}\left[\boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}-\widetilde{\boldsymbol{\beta}}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right]\right\}\\
&\propto |\boldsymbol{V}|^{-\frac{1}{2}}\left|\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right|^{1/2}\exp\left\{-\frac{1}{2}\left[\boldsymbol{y}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{y}-2\widetilde{\boldsymbol{\beta}}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\widetilde{\boldsymbol{\beta}}+\widetilde{\boldsymbol{\beta}}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right]\right\}\\
&\propto |\boldsymbol{V}|^{-\frac{1}{2}}\left|\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{X}\right|^{-1/2}\exp\left\{-\frac{1}{2}\left(\boldsymbol{y}-\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{V}^{-1}\left(\boldsymbol{y}-\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)\right\}
\end{aligned}$$

Appendix 4

R code for Example 1:

```
rm(list=ls())
library(sommer)
load('dat_ls_E1.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
#Marker data
dat_M = dat_ls$dat_M
dim(dat_M)
dat_F = transform(dat_F, GID = as.character(GID))
head(dat_F,5)
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
XM = dat_M[Pos,]
XM = scale(XM)
#Genomic relationship matrix derived from markers
dat_M = scale(dat_M)
G = tcrossprod(dat_M)/dim(dat_M)[2]
```

```
dat_F$GID = factor(dat_F$GID,levels=row.names(G))
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]

#10 random partitions
K = 10
n = dim(dat_F)[1]
set.seed(1)
PT = replicate(K,sample(n,0.20*n))

#Example 1
#GBLUP model
Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 dat_F$y_NA = dat_F$y
 dat_F$y_NA[Pos_tst] = NA
 #M1
 A = mmer(y_NA ~ 1,na.method.Y='include', random= ~ vs(GID,Gu=G),
          rcov= ~ vs(units), data=dat_F,verbose=FALSE)
 #BLUPs
 #bv = A$U$`u:GID`$y_NA
 yp = fitted(A)$dataWithFitted$y_NA.fitted
 #Prediction of testing
 yp_ts = yp[Pos_tst]
 #MSEP and Cor
 Tab$MSEP[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
 Tab$Cor[k] = cor(dat_F$y[Pos_tst],yp_ts)

 #M10
 A2 = mmer(y_NA ~ 1,na.method.Y='include', random= ~ vs(GID),
           rcov= ~ vs(units), data=dat_F,verbose=FALSE)
 yp2 = fitted(A2)$dataWithFitted$y_NA.fitted
 #Prediction of testing
 yp2_ts = yp2[Pos_tst]
 #MSEP
 Tab$MSEP10[k] = mean((dat_F$y[Pos_tst]-yp2_ts)^2)
 Tab$Cor10[k] = cor(dat_F$y[Pos_tst],yp2_ts)

}
```

Appendix 5

```
#Example 2
rm(list=ls())
library(sommer)
load('dat_ls_E1.RData',verbose=TRUE)
```

```
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
#Marker data
dat_M  = dat_ls$dat_M
dim(dat_M)
dat_F = transform(dat_F, GID = as.character(GID))
head(dat_F,5)
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
XM = dat_M[Pos,]
XM = scale(XM)
#Genomic relationship matrix derived from markers
dat_M = scale(dat_M)
G = tcrossprod(dat_M)/dim(dat_M)[2]

dat_F$GID = factor(dat_F$GID,levels=row.names(G))
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]

#10 random partitions
K = 10
n = dim(dat_F)[1]
set.seed(1)
PT = replicate(K,sample(n,0.20*n))

#Model (5.4)
#Y = mu + Env + GID + GID:Env
Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
dat_F$Env_GID = paste(dat_F$Env,dat_F$GID,sep='_')
GE = kronecker(diag(length(unique(dat_F$Env))),G)
rnGWE = expand.grid(row.names(G),unique(dat_F$Env))
row.names(GE) = paste(rnGWE[,2],rnGWE[,1],sep='_')
colnames(GE) = row.names(GE)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 dat_F$y_NA = dat_F$y
 dat_F$y_NA[Pos_tst] = NA
 #M2
 A = mmer(y_NA ~ Env,na.method.Y='include',
       random= ~ vs(GID,Gu=G)+ vs(Env_GID,Gu = GE),
       rcov= ~ vs(units),
       data=dat_F,verbose=FALSE)
 yp = fitted(A)$dataWithFitted$y_NA.fitted
 #Prediction of testing
 yp_ts = yp[Pos_tst]
 #MSEP
 Tab$MSEP[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
 Tab$Cor[k] = cor(dat_F$y[Pos_tst],yp_ts)

 #M20
 A20 = mmer(y_NA ~ Env,na.method.Y='include',
```

```
      random= ~ vs(GID)+vs(Env_GID),
      rcov= ~ vs(units),
      data=dat_F,verbose=FALSE)
 yp20 = fitted(A20)$dataWithFitted$y_NA.fitted
 #Prediction of testing
 yp20_ts = yp20[Pos_tst]
 Tab$MSEP20[k] = mean((dat_F$y[Pos_tst]-yp20_ts)^2)
 Tab$Cor20[k] = cor(dat_F$y[Pos_tst],yp20_ts)

 #M11
 A = mmer(y_NA ~ Env,na.method.Y='include',
       random= ~ vs(GID,Gu=G),
       rcov= ~ vs(units),
       data=dat_F,verbose=FALSE)
 yp = fitted(A)$dataWithFitted$y_NA.fitted
 #Prediction of testing
 yp_ts = yp[Pos_tst]
 Tab$MSEP11[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
 Tab$Cor11[k] = cor(dat_F$y[Pos_tst],yp_ts)
 #M10a: Model1 (5.4) plus environment effects but with G = I_J
 A = mmer(y_NA ~ Env,na.method.Y='include',
      random= ~ vs(GID),
      rcov= ~ vs(units),
      data=dat_F,verbose=FALSE)
 yp = fitted(A)$dataWithFitted$y_NA.fitted
 #Prediction of testing
 yp_ts = yp[Pos_tst]
 Tab$MSEP10a[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
 Tab$Cor10a[k] = cor(dat_F$y[Pos_tst],yp_ts)

}

#Basic code to implement model (5.4) with an unstructured form for
Sigma_E
A = mmer(y~ Env, na.method.Y='include',
         random= ~ vs(GID,Gu=G)+vs(us(Env),GID,Gu=G),
         rcov= ~ vs(units), data=dat_F)
#Basic code to implement model (5.4) with heterogeneous environment
variances
A = mmer(y ~ Env, ,
         random= ~ vs(GID,Gu=G)+
                   vs(ds(Env),GID,Gu=G),#or vs(at(Env),GID,Gu=G)
         rcov= ~ vs(units), data=dat_F)
#Y = mu + Env + GID + GID:ENV
#Y = mu + Env + GID + GID:ENV
#Basic code to implement model (5.4) with heterogeneous environment and
residuals variances
A = mmer(y ~ Env,
     random= ~ vs(GID,Gu=G)+ vs(ds(Env),GID,Gu=G),
     rcov= ~ vs(ds(Env),units),
     data=dat_F)
```

Appendix 6

```
rm(list=ls())
library(sommer)
load('dat_ls_E1.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
#Marker data
dat_M = dat_ls$dat_M
dim(dat_M)
dat_F = transform(dat_F, GID = as.character(GID))
head(dat_F,5)
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
XM = dat_M[Pos,]
XM = scale(XM)
dim(XM)
n = dim(dat_F)[1]
#Genomic relationship matrix derived from markers
dat_M = scale(dat_M)
G = tcrossprod(dat_M)/dim(dat_M)[2]

dat_F$GID = factor(dat_F$GID,levels=row.names(G))
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]

#10 random partitions
K = 10
set.seed(1)
PT = replicate(K,sample(n,0.20*n))



#Example 3
Tab = data.frame(PT = 1:K)
set.seed(1)
for(k in 1:K)
{
 #M3
 Pos_tst = PT[,k]
 dat_F$y_NA = dat_F$y
 dat_F$y_NA[Pos_tst] = NA
 dat_F$y2_NA = dat_F$y2
 dat_F$y2_NA[Pos_tst] = NA
 A = mmer(cbind(y_NA,y2_NA) ~ 1,na.method.Y='include',
      random= ~ vs(GID,Gu=G) ,
      rcov= ~ vs(units),
      data=dat_F,verbose=FALSE)
 #BLUPs
 b_ls = A$U$`u:GID`
 b_T1 = b_ls$y_NA# Trait 1
 b_T2 = b_ls$y2_NA# Trait 2
```

```
b_mat = cbind(b_T1,b_T2)
Pos = match(dat_F$GID,names(b_ls$y_NA))
#Y "fitted"
yp = A$fitted + b_mat[Pos,]
#Prediction of testing for both traits
yp_ts = yp[Pos_tst,1]# Trait 1
y2p_ts = yp[Pos_tst,2]# Trait 2
#MSEP and Cor
#Trait 1
Tab$MSEP_T1[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
Tab$Cor_T1[k] = cor(dat_F$y[Pos_tst],yp_ts)
#Trait 2
Tab$MSEP_T2[k] = mean((dat_F$y2[Pos_tst]-y2p_ts)^2)
Tab$Cor_T2[k] = cor(dat_F$y2[Pos_tst],y2p_ts)
#M32: Sigma_T diagonal
A = mmer(cbind(y_NA,y2_NA) ~ 1,na.method.Y='include',
     random= ~ vs(GID,Gu=G,Gtc=diag(2)),
     rcov= ~ vs(units),
     data=dat_F,verbose=FALSE)
#BLUPs
b_ls = A$U$`u:GID`
b_T1 = b_ls$y_NA# Trait 1
b_T2 = b_ls$y2_NA# Trait 2
b_mat = cbind(b_T1,b_T2)
Pos = match(dat_F$GID,names(b_ls$y_NA))
#Y "fitted"
yp = A$fitted + b_mat[Pos,]
#Prediction of testing
yp_ts = yp[Pos_tst,1]# Trait 1
y2p_ts = yp[Pos_tst,2]# Trait 2
#MSEP and Cor
#Trait 1
Tab$MSEP_T1_32[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
Tab$Cor_T1_32[k] = cor(dat_F$y[Pos_tst],yp_ts)
#Trait 2
Tab$MSEP_T2_32[k] = mean((dat_F$y2[Pos_tst]-y2p_ts)^2)
Tab$Cor_T2_32[k] = cor(dat_F$y2[Pos_tst],y2p_ts)

# #M3
# #dat_F$y2_NA = dat_F$y2
# dat_F$y_NA = dat_F$y# Trait T1 is not NA
# A = mmer(cbind(y_NA,y2_NA) ~ 1,na.method.Y='include',
#      random= ~ vs(GID,Gu=G),
#      rcov= ~ vs(units),#tolparinv = 1e-2,
#      data=dat_F,verbose=FALSE)
# #BLUPs
# b_ls = A$U$`u:GID`
# b_T1 = b_ls$y_NA# Trait 1
# b_T2 = b_ls$y2_NA# Trait 2
# b_mat = cbind(b_T1,b_T2)
# Pos = match(dat_F$GID,names(b_ls$y_NA))
# #Y "fitted"
# yp = A$fitted + b_mat[Pos,]
```

```
 # #Prediction of testing
 # yp_ts = yp[Pos_tst,1]# Trait 1
 # y2p_ts = yp[Pos_tst,2]# Trait 2
 # #MSEP
 # Tab$MSEP_T1_31[k] = mean((dat_F$y[Pos_tst]-yp_ts)^2)
 # Tab$Cor_T1_31[k] = cor(dat_F$y[Pos_tst],yp_ts)
 # Tab$MSEP_T2_31[k] = mean((dat_F$y2[Pos_tst]-y2p_ts)^2)
 # Tab$Cor_T2_31[k] = cor(dat_F$y2[Pos_tst],y2p_ts)
 }
```

Appendix 7

```
#Example 4
rm(list=ls())
library(sommer)
load('dat_ls_E4.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
dat_F = transform(dat_F, GID = as.character(GID))
#Genomic relationship matrix derived from markers
G = dat_ls$G
dat_F$GID = factor(dat_F$GID,levels=row.names(G))
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
#Fitting model M4 with data set
#A = mmer(cbind(GY,TESTWT) ~ Env,na.method.Y='include',
#      random= ~ vs(GID,Gu=G) + vs(Env_GID,Gu=GE) ,
#      rcov= ~ vs(units),
#      data=dat_F,verbose=FALSE)
#Example 4
#10 random partitions
n = dim(dat_F)[1] ;K = 10
set.seed(1)
PT = replicate(K,sample(n,0.20*n))
#Model 5.6
#Y = mu + Env + GID + GID:Env + e
Tab = data.frame(PT = 1:K)
set.seed(1)
dat_F$Env_GID = paste(dat_F$Env,dat_F$GID,sep='_')
GE = kronecker(diag(length(unique(dat_F$Env))),G)
rnGWE = expand.grid(row.names(G),unique(dat_F$Env))
row.names(GE) = paste(rnGWE[,2],rnGWE[,1],sep='_')
colnames(GE) = row.names(GE)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 #M4
 Pos_tst = PT[,k]
 dat_F$GY_NA = dat_F$GY
 dat_F$GY_NA[Pos_tst] = NA
```

```
dat_F$TESTWT_NA = dat_F$TESTWT
dat_F$TESTWT_NA[Pos_tst] = NA
A = mmer(cbind(GY_NA,TESTWT_NA ) ~ Env,na.method.Y='include',
      random= ~ vs(GID,Gu=G) + vs(Env_GID,Gu=GE) ,
      rcov= ~ vs(units),tolparinv = 1,
      data=dat_F,verbose=FALSE)
#BLUPs
b_ls = A$U$`u:GID`
b_T1 = b_ls$GY_NA# Trait 1
b_T2 = b_ls$TESTWT_NA# Trait 2
b_mat = cbind(b_T1,b_T2)
Pos = match(dat_F$GID,names(b_ls$TESTWT))
#Y "fitted"
yp = data.frame(A$fitted + b_mat[Pos,] )
colnames(yp) = names(b_ls)
plot(dat_F$GY,yp$GY_NA);abline(a=0,b=1)
#Prediction of testing of both traits
yp_tst = yp[Pos_tst,]
#plot(dat_F$y2,yp[,2]); abline(a=0,b=1)
#MSEP and Cor
#Trait 1
Tab$MSEPT1[k] = mean((dat_F$GY[Pos_tst]-yp_tst$GY_NA)^2)
Tab$CorT1[k] = cor(dat_F$GY[Pos_tst],yp_tst$TESTWT_NA)
#Trait 2
Tab$MSEPT2[k] = mean((dat_F$TESTWT[Pos_tst]-yp_tst$TESTWT_NA)^2)
Tab$CorT2[k] = cor(dat_F$TESTWT[Pos_tst],yp_tst$TESTWT_NA)

#M42
A42 = mmer(cbind(GY_NA,TESTWT_NA) ~ Env,na.method.Y='include',
      random= ~ vs(GID,Gu=G) + vs(Env_GID,Gu=GE,Gtc=diag(2)) ,
      rcov= ~ vs(units),
      data=dat_F,verbose=FALSE)
#BLUPs
b_ls = A42$U$`u:GID`
b_T1 = b_ls$GY_NA# Trait 1
b_T2 = b_ls$TESTWT_NA# Trait 2
b_mat = cbind(b_T1,b_T2)
Pos = match(dat_F$GID,names(b_ls$TESTWT))
#Y "fitted"
yp = data.frame(A$fitted + b_mat[Pos,] )
colnames(yp) = names(b_ls)
#plot(dat_F$GY,yp$GY_NA);abline(a=0,b=1)
#Prediction of testing of both traits
yp_tst = yp[Pos_tst,]
#MSEP and Cor
#Trait 1
Tab$MSEPT1_42[k] = mean((dat_F$GY[Pos_tst]-yp_tst$GY_NA)^2)
Tab$CorT1_42[k] = cor(dat_F$GY[Pos_tst],yp_tst$TESTWT_NA)
#Trait 2
Tab$MSEPT2_42[k] = mean((dat_F$TESTWT[Pos_tst]-yp_tst$TESTWT_NA)
^2)
Tab$CorT2_42[k] = cor(dat_F$TESTWT[Pos_tst],yp_tst$TESTWT_NA)
```

```
#M43
A43 = mmer(cbind(GY_NA,TESTWT_NA) ~ Env,na.method.Y='include',
      random= ~ vs(GID,Gu=G,Gtc=diag(2)) +
           vs(Env_GID,Gu=GE,Gtc=diag(2)) ,
      rcov= ~ vs(units),
      data=dat_F,verbose=FALSE)
#BLUPs
b_ls = A43$U$`u:GID`
b_T1 = b_ls$GY_NA# Trait 1
b_T2 = b_ls$TESTWT_NA# Trait 2
b_mat = cbind(b_T1,b_T2)
Pos = match(dat_F$GID,names(b_ls$TESTWT))
#Y "fitted"
yp = data.frame(A$fitted + b_mat[Pos,] )
colnames(yp) = names(b_ls)
#plot(dat_F$GY,yp$GY_NA);abline(a=0,b=1)
#Prediction of testing of both traits
yp_tst = yp[Pos_tst,]
#MSEP and Cor
#Trait 1
Tab$MSEPT1_43[k] = mean((dat_F$GY[Pos_tst]-yp_tst$GY_NA)^2)
Tab$CorT1_43[k] = cor(dat_F$GY[Pos_tst],yp_tst$TESTWT_NA)
#Trait 2
Tab$MSEPT2_43[k] = mean((dat_F$TESTWT[Pos_tst]-yp_tst$TESTWT_NA)
^2)
Tab$CorT2_43[k] = cor(dat_F$TESTWT[Pos_tst],yp_tst$TESTWT_NA)
 A44 = mmer(cbind(GY_NA,TESTWT_NA) ~ Env,na.method.Y='include',
      random= ~ vs(GID,Gu=G,Gtc=diag(2)) +
       vs(Env_GID,Gu=GE,Gtc=diag(2)) ,
      rcov= ~ vs(units,Gtc=diag(2)),
      data=dat_F,verbose=FALSE)
#A44$sigma
#BLUPs
b_ls = A44$U$`u:GID`
b_T1 = b_ls$GY_NA# Trait 1
b_T2 = b_ls$TESTWT_NA# Trait 2
b_mat = cbind(b_T1,b_T2)
Pos = match(dat_F$GID,names(b_ls$TESTWT))
#Y "fitted"
yp = data.frame(A$fitted + b_mat[Pos,] )
colnames(yp) = names(b_ls)
#plot(dat_F$GY,yp$GY_NA);abline(a=0,b=1)
#Prediction of testing of both traits
yp_tst = yp[Pos_tst,]
#MSEP and Cor
#Trait 1
Tab$MSEPT1_44[k] = mean((dat_F$GY[Pos_tst]-yp_tst$GY_NA)^2)
Tab$CorT1_44[k] = cor(dat_F$GY[Pos_tst],yp_tst$TESTWT_NA)
#Trait 2
Tab$MSEPT2_44[k] = mean((dat_F$TESTWT[Pos_tst]-yp_tst$TESTWT_NA)
^2)
Tab$CorT2_44[k] = cor(dat_F$TESTWT[Pos_tst],yp_tst$TESTWT_NA)
cat('k=',k,'\n')

}
```

```
#Model 5.6 with Sigma_2T different for each Env
A4 = mmer(cbind(GY,TESTWT)~Env,
      random = ~vs(GID,Gu=G)+vs(ds(Env),GID,Gu=G),data=dat_F,
      rcov= ~ vs(units))
```

References

Araus JL, Cairns JE (2014) Field high-throughput phenotyping: the new crop breeding frontier. Trends Plant Sci 19(1):52–61

Bates D, Maechler M, Bolker B, Walker S (2015) Fitting linear mixed-effects models using lme4. J Stat Softw 67(1):1–48

Borman S (2004) The expectation maximization algorithm: a short tutorial. https://www.lri.fr/~sebag/COURS/EM_algorithm.pdf

Brown H, Prescott R (2014) Applied mixed models in medicine. John Wiley & Sons, Hoboken, NJ

Budhlakoti N, Mishra DC, Rai A, Lal SB, Chaturvedi KK, Kumar RR (2019) A comparative study of single-trait and multi-trait genomic selection. J Comput Biol 26(10):1100–1112

Burgueño J, de los Campos G, Weigel K, Crossa J (2012) Genomic prediction of breeding values when modeling genotype × environment interaction using pedigree and dense molecular markers. Crop Sci 52(2):707–719

Cabrera-Bosquet L, Crossa J, von Zitzewitz J, Serret MD, Luis Araus J (2012) High-throughput phenotyping and genomic selection: the frontiers of crop breeding converge F. J Integr Plant Biol 54(5):312–320

Calus MP, Veerkamp RF (2011) Accuracy of multi-trait genomic selection using different methods. Genet Select Evol 43(1):26. https://doi.org/10.1186/1297-9686-43-26

Cappa EP, de Lima BM, da Silva-Junior OB, Garcia CC, Mansfield SD, Grattapaglia D (2019) Improving genomic prediction of growth and wood traits in Eucalyptus using phenotypes from non-genotyped trees by single-step GBLUP. Plant Sci 284:9–15

Covarrubias-Pazaran G (2016) Genome-assisted prediction of quantitative traits using the R package sommer. PLoS One 11(6):e0156744

Covarrubias-Pazaran G (2018) Software update: moving the R package sommer to multivariate mixed models for genome-assisted prediction. https://doi.org/10.1101/354639

Covarrubias-Pazaran G, Schlautman B, Diaz-Garcia L, Grygleski E, Polashock J, Johnson-Cicalese J et al (2018) Multivariate GBLUP improves accuracy of genomic selection for yield and fruit weight in biparental populations of Vaccinium macrocarpon Ait. Front Plant Sci 9:1310

Crossa J, Pérez-Rodríguez P, Cuevas J, Montesinos-López O, Jarquín D, de los Campos G et al (2017) Genomic selection in plant breeding: methods, models, and perspectives. Trends Plant Sci 22(11):961–975

Finch WH, Bolin JE, Kelley K (2019) Multilevel modeling using R. CRC Press, Boca Raton, FL

Ghosh S, Watson A, Gonzalez-Navarro OE, Ramirez-Gonzalez RH, Yanes L, Mendoza-Suárez M et al (2018) Speed breeding in growth chambers and glasshouses for crop breeding and model plant research. Nat Protoc 13(12):2944–2963

Goldstein H (2011) Multilevel statistical models. Wiley, Hoboken, NJ

Harville DA (1974) Bayesian inference for variance components using only error contrasts. Biometrika 61(2):383–385

Harville DA (1977) Maximum likelihood approaches to variance component estimation and to related problems. J Am Stat Assoc 72(358):320–338

Harville DA (1985) Decomposition of prediction error. J Am Stat Assoc 80(389):132–138

Henderson CR (1950) Estimation of genetic parameters. Ann Math Stat 21:309–310

Henderson CR (1975) Best linear unbiased estimation and prediction under a selection model. Biometrics 31:423–447

Henderson CR, Quaas RL (1976) Multiple trait evaluation using relatives' records. J Anim Sci 43 (6):1188–1197

Jarquín D, Crossa J, Lacaze X, Du Cheyron P, Daucourt J, Lorgeou J et al (2014) A reaction norm model for genomic selection using high-dimensional genomic and environmental data. Theor Appl Genet 127(3):595–607
Jennrich RI, Sampson PF (1976) Newton-Raphson and related algorithms for maximum likelihood variance component estimation. Technometrics 18(1):11–17
Jennrich RI, Schluchter MD (1986) Unbalanced repeated-measures models with structured covariance matrices. Biometrics 42:805–820
Jia Y, Jannink JL (2012) Multiple-trait genomic selection methods increase genetic value prediction accuracy. Genetics 192(4):1513–1522
Jiang J, Zhang Q, Ma L, Li J, Wang Z, Liu JF (2015) Joint prediction of multiple quantitative traits using a Bayesian multivariate antedependence model. Heredity 115(1):29–36
Johnson RA, Wichern DW (2002) Applied multivariate statistical analysis. Prentice Hall, Upper Saddle River, NJ
Krause MR, González-Pérez L, Crossa J, Pérez-Rodríguez P, Montesinos-López O, Singh RP et al (2019) Hyperspectral reflectance-derived relationship matrices for genomic prediction of grain yield in wheat. G3 9(4):1231–1247
Laird NM, Ware JH (1982) Random-effects models for longitudinal data. Biometrics 38:963–974
Leyland AH, Goldstein H (2001) Multilevel modelling of health statistics. Wiley, Hoboken, NJ
Lindstrom MJ, Bates DM (1988) Newton–Raphson and EM algorithms for linear mixed-effects models for repeated-measures data. J Am Stat Assoc 83(404):1014–1022
Lorenz AJ, Smith KP (2015) Adding genetically distant individuals to training populations reduces genomic prediction accuracy in barley. Crop Sci 55(6):2657–2667
Martini JW, Crossa J, Toledo FH, Cuevas J (2020) On Hadamard and Kronecker products in covariance structures for genotype × environment interaction. Plant Genome 13:e20033
McLean RA, Sanders WL, Stroup WW (1991) A unified approach to mixed linear models. Am Stat 45(1):54–64
Meeker W, Hong Y, Escobar L (2011) Degradation models and analyses. In: Encyclopedia of statistical sciences. Wiley, Hoboken, NJ. https://doi.org/10.1002/0471667196.ess7148
Meuwissen THE, Hayes BJ, Goddard ME (2001) Prediction of total genetic values using genome-wide dense marker maps. Genetics 157:1819–1829
Montesinos-López OA, Montesinos-López A, Crossa J, Toledo FH, Pérez-Hernández O, Eskridge KM, Rutkoski J (2016) A genomic Bayesian multi-trait and multi-environment model. G3 6 (9):2725–2744
Muranty H, Troggio M, Sadok IB, Al Rifaï M, Auwerkerken A, Banchi E et al (2015) Accuracy and responses of genomic selection on key traits in apple breeding. Horticult Res 2(1):1–12
Okeke UG, Akdemir D, Rabbi I, Kulakow P, Jannink JL (2017) Accuracies of univariate and multivariate genomic prediction models in African cassava. Genet Sel Evol 49(1):88
Patterson HD, Thompson R (1971) Recovery of inter-block information when block sizes are unequal. Biometrika 58:545–554
Piepho HP, Möhring J, Melchinger AE, Büchse A (2008) BLUP for phenotypic selection in plant breeding and variety testing. Euphytica 161(1–2):209–228
Pinheiro JC, Bates DM (2000) Mixed-effects models in S and S-PLUS. Springer, New York
Poland J, Endelman J, Dawson J, Rutkoski J, Wu S, Manes Y et al (2012) Genomic selection in wheat breeding using genotyping-by-sequencing. Plant Genome 5(3):103–113
Raudenbush SW, Bryk AS (2002) Hierarchical linear models: applications and data analysis methods. Sage Publications, Inc, Thousand Oaks, CA
Rencher AC (2008) Linear models in statistics. Wiley, Hoboken, NJ
Robinson GK (1991) That BLUP is a good thing: the estimation of random effects. Stat Sci 6 (1):15–32
Runcie D, Cheng H (2019) Pitfalls and remedies for cross validation with multi-trait genomic prediction methods. G3 9(11):3727–3741

Searle SR (1993) Applying the EM algorithm to calculating ML and REML estimates of variance components. In: Paper invited for the 1993 American Statistical Association Meeting, San Francisco
Searle SR, Casella G, McCulloch CE (2006) Variance components. Wiley, Hoboken, NJ
Speelman D, Heylen K, Geeraerts D (eds) (2018) Mixed-effects regression models in linguistics. Springer, New York
Stroup WW (2012) Generalized linear mixed models: modern concepts, methods and applications. CRC Press, Boca Raton, FL
VanRaden PM (2008) Efficient methods to compute genomic predictions. J Dairy Sci 91:4414–4423
Wang X, Xu Y, Hu Z, Xu C (2018) Genomic selection methods for crop improvement: current status and prospects. Crop J 6(4):330–340
Watson A, Hickey LT, Christopher J, Rutkoski J, Poland J, Hayes BJ (2019) Multivariate genomic selection and potential of rapid indirect selection with speed breeding in spring wheat. Crop Sci 59(5):1945–1959
West BT, Welch KB, Galecki AT (2014) Linear mixed models: a practical guide using statistical software. CRC Press, Boca Raton, FL
Zuur A, Ieno EN, Walker N, Saveliev AA, Smith GM (2009) Mixed effects models and extensions in ecology with R. Springer Science & Business Media, New York

Chapter 6
Bayesian Genomic Linear Regression

6.1 Bayes Theorem and Bayesian Linear Regression

Unlike classic statistical inference, which assumes that the parameter $\boldsymbol{\theta}$ that defines the model is a fixed unknown quantity, in Bayesian inference it is considered as a random variable whose variation tries to represent the knowledge or ignorance about this, before the data points are collected (Box and Tiao 1992). The probability density function which describes such variation is known as the prior distribution and is an additional component in the specification of the complete model in a Bayesian framework.

Given a data set $\boldsymbol{y}_n = (y_1, \ldots, y_n)$ whose distribution is assumed to be $f(\boldsymbol{y}|\boldsymbol{\theta})$, and a prior distribution for the parameter $\boldsymbol{\theta}$, $f(\boldsymbol{\theta})$, the Bayesian analysis uses the Bayes theorem to combine these two pieces of information to obtain the posterior distribution of the parameters, on which the inference is fully based (Christensen et al. 2011):

$$f(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{f(\boldsymbol{y},\boldsymbol{\theta})}{f(\boldsymbol{y})} = \frac{f(\boldsymbol{\theta})f(\boldsymbol{y}|\boldsymbol{\theta})}{f(\boldsymbol{y})} \propto f(\boldsymbol{\theta})L(\boldsymbol{\theta};\boldsymbol{y}),$$

where $f(\boldsymbol{y}) = \int f(\boldsymbol{y}|\boldsymbol{\theta})f(\boldsymbol{\theta})d\boldsymbol{\theta} = E_{\boldsymbol{\theta}}[f(\boldsymbol{y}|\boldsymbol{\theta})]$ is the marginal distribution of $\boldsymbol{\theta}$. This conditional distribution describes what is known about $\boldsymbol{\theta}$ after data is collected and can be thought of as the updated prior knowledge about $\boldsymbol{\theta}$ with the information contained in the data, which is done through the likelihood function $L(\boldsymbol{\theta};\boldsymbol{y})$ (Box and Tiao 1992).

In general, because the posterior distribution doesn't always have a recognizable form and it is often not easy to simulate from this, numerical approximation methods are employed. Once a sample of the posterior distribution is obtained, estimation of a parameter is often found by averaging the sample values or averaging a function of the sample values when another quantity is of interest. For example, in genomic prediction with dense molecular markers, the main interest is to predict the trait of

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_6

interest of the non-phenotyped individuals that have only genotypic information, environment variables, or other information (covariates). In this situation, a convenient practice is to include the individuals to be predicted ($\boldsymbol{y}_p$) in the posterior distribution to be sampled.

Specifically, a standard Bayesian framework for a normal linear regression model (see Chap. 3)

$$Y = \beta_0 + \sum_{j=1}^{p} X_j \beta_j + \epsilon \tag{6.1}$$

with ϵ a random error with normal distribution with mean 0 and variance σ^2, is fully specified by assuming the next non-informative prior distribution: $\boldsymbol{\beta}$ and $\log(\sigma)$ approximately independent and locally uniform.

$$f(\boldsymbol{\beta}, \sigma^2) \propto \sigma^{-2} \tag{6.2}$$

which is not a proper distribution because it does not integrate to 1 (Box and Tiao 1992; Gelman et al. 2013). However, when $\boldsymbol{X}$ is of full column rank, the posterior distribution is a proper distribution and is given by

$$\begin{aligned}
f(\boldsymbol{\beta},\sigma^2|\boldsymbol{y},\boldsymbol{X}) &\propto (\sigma^2)^{-\frac{n}{2}} \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})\right](\sigma^2)^{-1} \\
&\propto (\sigma^2)^{-\frac{n}{2}} \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y})\right](\sigma^2)^{-1} \\
&\propto (\sigma^2)^{-\frac{p+1}{2}} \exp\left[-\frac{1}{2\sigma^2}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right) - \frac{1}{2\sigma^2}\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \widetilde{\boldsymbol{\beta}}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\widetilde{\boldsymbol{\beta}}\right)\right](\sigma^2)^{-1-(n-p-1)/2} \\
&\propto (\sigma^2)^{-\frac{p+1}{2}} \exp\left[-\frac{1}{2\sigma^2}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)\right](\sigma^2)^{-1-(n-p-1)/2} \exp\left[-\frac{1}{2\sigma^2}\boldsymbol{y}^{\mathrm{T}}(\boldsymbol{I}-\boldsymbol{H})\boldsymbol{y}\right] \\
&\propto (\sigma^2)^{-\frac{p+1}{2}} \exp\left[-\frac{1}{2\sigma^2}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\left(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}\right)\right](\sigma^2)^{-1-\frac{n-p-1}{2}} \exp\left(-\frac{(n-p-1)\widetilde{\sigma}^2}{2\sigma^2}\right),
\end{aligned}$$

where $\widetilde{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$, $\widetilde{\sigma}^2 = \frac{1}{n-p-1}\boldsymbol{y}^{\mathrm{T}}(\boldsymbol{I}-\boldsymbol{H})\boldsymbol{y}$, and $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}$. From here the marginal posterior distribution of σ^2 is $\sigma^2 \mid \boldsymbol{y}, \boldsymbol{X} \sim IG\left((n-p-1)/2, \frac{(n-p-1)\widetilde{\sigma}^2}{2}\right)$ with mean $\frac{(n-p-1)\widetilde{\sigma}^2}{2}/[(n-p-1)/2] = \widetilde{\sigma}^2$, and given σ^2, the posterior conditional distribution of $\boldsymbol{\beta}$ is given by $\boldsymbol{\beta} \mid \sigma^2, \boldsymbol{y}, \boldsymbol{X} \sim N\left(\widetilde{\boldsymbol{\beta}}, \sigma^2\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\right)$.

6.2 Bayesian Genome-Based Ridge Regression

When $p > n$, $\boldsymbol{X}$ is not of full column rank and the posterior of model (6.1) may not be proper (Gelman et al. 2013), so a solution is instead to consider independently proper prior distributions, $\boldsymbol{\beta} \sim N(0, \boldsymbol{I}\sigma^2)$ and $\sigma^2 \sim IG(\alpha_0, \alpha_0)$, which for large values of σ^2

(10^6) and small values of α_0 (10^{-3}) is an approximation to the standard non-informative prior given in (6.1) (Christensen et al. 2011). A similar prior specification is taken in genomic prediction where different models are obtained by adopting different prior distributions of the parameters. For example, the Bayesian Linear Ridge Regression (Pérez and de los Campos 2014) with standardized covariates ($X_j's$) is given by

$$Y = \mu + \sum_{j=1}^{p} X_j \beta_j + \epsilon \tag{6.3}$$

with a flat prior for mean parameter (μ), $f(\mu) \propto 1$, which can be approximately specified by $\mu \sim N(0, \sigma_0^2)$, with a large value of σ_0^2 (10^{10}), a multivariate normal distribution with mean vector $\mathbf{0}$ and covariance matrix $\sigma_\beta^2 \boldsymbol{I}_p$ on the beta coefficients, $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_p)^{\mathrm{T}} \mid \sigma_\beta^2 \sim N_p\left(\mathbf{0}, \boldsymbol{I}_p \sigma_\beta^2\right)$, and scaled inverse Chi-square distributions as priors for the variance component: $\sigma_\beta^2 \sim \chi^{-2}_{v_\beta,\ S_\beta}$ (prior for the variance of the regression coefficients β_j) and $\sigma^2 \sim \chi^{-2}_{v,S}$ (prior for the variance of random errors, ϵ), where $\chi^{-2}_{v,S}$ denotes the scaled inverse Chi-squared distribution with shape parameter v and scale parameter S. The joint posterior distribution of the parameters in this model, $\boldsymbol{\theta} = \left(\mu, \boldsymbol{\beta}_0^{\mathrm{T}}, \sigma^2, \sigma_\beta^2\right)^{\mathrm{T}}$, is given by

$$\begin{aligned}
f\left(\mu, \boldsymbol{\beta}_0^{\mathrm{T}}, \sigma_\beta^2, \sigma^2 | \boldsymbol{y}, \boldsymbol{X}\right) &\propto L\left(\mu, \boldsymbol{\beta}_0^{\mathrm{T}}, \sigma^2; \boldsymbol{y}\right) f(\boldsymbol{\theta}) \\
&\propto L\left(\mu, \boldsymbol{\beta}_0^{\mathrm{T}}, \sigma^2; \boldsymbol{y}\right) f(\mu) f\left(\boldsymbol{\beta}_0 | \sigma_\beta^2\right) f\left(\sigma_\beta^2\right) f\left(\sigma^2\right) \\
&\propto \frac{1}{\left(2\pi\sigma^2\right)^{\frac{n}{2}}} \exp\left[-\frac{1}{2\sigma^2} \|\boldsymbol{y} - 1_n \mu - \boldsymbol{X}_1 \boldsymbol{\beta}_0\|^2\right] \times \exp\left(-\frac{1}{2\sigma_0^2} \mu^2\right) \\
&\times \frac{1}{\left(\sigma_\beta^2\right)^{\frac{p}{2}}} \exp\left(\left[-\frac{1}{2\sigma_\beta^2} \boldsymbol{\beta}_0^{\mathrm{T}} \boldsymbol{\beta}_0\right]\right) \times \frac{\left(\frac{S_\beta}{2}\right)^{\frac{v_\beta}{2}}}{\Gamma\left(\frac{v_\beta}{2}\right)\left(\sigma_\beta^2\right)^{1+\frac{v_\beta}{2}}} \exp\left(-\frac{S_\beta}{2\sigma_\beta^2}\right) \\
&\times \frac{\left(\frac{S}{2}\right)^{\frac{v}{2}}}{\Gamma\left(\frac{v}{2}\right)\left(\sigma^2\right)^{1+\frac{v}{2}}} \exp\left(-\frac{S}{2\sigma^2}\right).
\end{aligned}$$

This has no known form and it is not easy to simulate values of it, so numerical methods are required to explore it. One way to simulate values of this distribution is by means of the Gibbs sampler method, which consists of alternatingly generating samples of the full conditional distributions of each variable (or block of variables) given the rest of the parameters (Casella and George 1992).

The full conditional posteriors to implement the Gibbs sampler are obtained in the lines below.

The conditional posterior distribution of $\boldsymbol{\beta}_0$ is given by

$$\begin{aligned} f(\boldsymbol{\beta}_0|-) &\propto L(\mu, \boldsymbol{\beta}_0^{\mathrm{T}}, \sigma^2; \boldsymbol{y}) f\left(\boldsymbol{\beta}_0|\sigma_\beta^2\right) \\ &\propto \exp\left[-\frac{1}{2\sigma^2}\|\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_1\boldsymbol{\beta}_0\|^2 - \frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}_0^{\mathrm{T}}\boldsymbol{\beta}_0\right] \\ &\propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{\beta}_0^{\mathrm{T}}\left(\sigma_\beta^{-2}\boldsymbol{I}_n + \sigma^{-2}\boldsymbol{X}_1^{\mathrm{T}}\boldsymbol{X}_1\right)\boldsymbol{\beta}_0 - 2\sigma^{-2}(\boldsymbol{y} - 1_n\mu)^{\mathrm{T}}\boldsymbol{X}_1\boldsymbol{\beta}_0\right]\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left[\left(\boldsymbol{\beta}_0 - \widetilde{\boldsymbol{\beta}}_0\right)^{\mathrm{T}}\widetilde{\boldsymbol{\Sigma}}_0^{-1}\left(\boldsymbol{\beta}_0 - \widetilde{\boldsymbol{\beta}}_0\right)\right]\right\} \end{aligned}$$

$\widetilde{\boldsymbol{\Sigma}}_0 = \left(\sigma_\beta^{-2}\boldsymbol{I}_p + \sigma^{-2}\boldsymbol{X}_1^{\mathrm{T}}\boldsymbol{X}_1\right)^{-1}$ and $\widetilde{\boldsymbol{\beta}}_0 = \sigma^{-2}\widetilde{\boldsymbol{\Sigma}}_0\boldsymbol{X}_1^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{1}_n\mu)$. That is, $\boldsymbol{\beta}_0 \mid - \sim \boldsymbol{N}_p\left(\widetilde{\boldsymbol{\beta}}_0, \widetilde{\boldsymbol{\Sigma}}_0\right)$. Similarly, the conditional distribution of μ is $\mu \mid - \sim \boldsymbol{N}(\widetilde{\mu}, \widetilde{\sigma}_0^2)$, where $\widetilde{\sigma}_0^2 = \frac{\sigma^2}{n}$ and $\widetilde{\mu} = \frac{1}{n}\boldsymbol{1}_n^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_0)$.

The conditional distribution of σ^2 is

$$\begin{aligned} f\left(\sigma^2|-\right) &\propto L(\mu, \boldsymbol{\beta}_0^{\mathrm{T}}, \sigma^2; \boldsymbol{y}) f\left(\sigma^2\right) \\ &\propto \frac{1}{(\sigma^2)^{\frac{n}{2}}}\exp\left[-\frac{1}{2\sigma^2}\|\boldsymbol{y} - 1_n\mu - \boldsymbol{X}_1\boldsymbol{\beta}_0\|^2\right]\frac{\left(\frac{S}{2}\right)^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)(\sigma^2)^{1+\frac{\nu}{2}}}\exp\left(-\frac{S}{2\sigma^2}\right) \\ &\propto \frac{\left(\frac{\widetilde{S}}{2}\right)^{\frac{\widetilde{\nu}}{2}}}{(\sigma^2)^{1+\frac{\widetilde{\nu}}{2}}}\exp\left(-\frac{\widetilde{S}}{2\sigma^2}\right), \end{aligned}$$

where $\widetilde{\nu} = \nu + n$ and $\widetilde{S} = S + ||\boldsymbol{y} - \boldsymbol{1}_n\mu - \boldsymbol{X}_1\boldsymbol{\beta}_0||^2$. So $\sigma^2 \mid - \sim \chi^{-2}_{\widetilde{\nu},\widetilde{S}}$, where $\chi^{-2}_{\nu,s}$ denotes a scaled inverse Chi-squared distribution with parameters ν and S. Similarly, $\sigma_\beta^2 \mid - \sim \chi^{-2}_{\widetilde{\nu}_\beta,\widetilde{S}_\beta}$, where $\widetilde{\nu}_\beta = \nu_\beta + p$ and $\widetilde{S}_\beta = S_\beta + \boldsymbol{\beta}_0^{\mathrm{T}}\boldsymbol{\beta}_0$.

In summary, for the Ridge regression model, a Gibbs sampler consists of the following steps:

1. Choose initial values for μ, $\boldsymbol{\beta}_0$, and σ^2.
2. Simulate a value of the full conditional distribution of σ_β^2:

$$\sigma_\beta^2 \mid \mu, \boldsymbol{\beta}_0, \sigma^2 \sim \chi^{-2}_{\widetilde{\nu},S_\beta},$$

where $\chi^{-2}_{\widetilde{\nu},S_\beta}$ denotes a scaled inverse Chi-square distribution with shape parameter $\widetilde{\nu}_\beta = \nu_\beta + p$ and scale parameter $\widetilde{S}_\beta = S_\beta + \boldsymbol{\beta}_0^{\mathrm{T}}\boldsymbol{\beta}_0$.

3. Simulate the full conditional posterior distribution of β_0:

$$\boldsymbol{\beta}_0 \mid \mu, \sigma_\beta^2, \sigma^2 \sim \boldsymbol{N}_p\left(\widetilde{\boldsymbol{\beta}}_0, \widetilde{\boldsymbol{\Sigma}}_0\right),$$

where $\widetilde{\boldsymbol{\Sigma}}_0 = \left(\sigma_\beta^{-2}\boldsymbol{I}_p + \sigma^{-2}\boldsymbol{X}_1^{\mathrm{T}}\boldsymbol{X}_1\right)^{-1}$ and $\widetilde{\boldsymbol{\beta}}_0 = \sigma^{-2}\widetilde{\boldsymbol{\Sigma}}_0\boldsymbol{X}_1^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{1}_n\mu)$

4. Simulate the full conditional distribution of μ:

$$\mu \mid \boldsymbol{\beta}_0, \sigma_\beta^2, \sigma^2 \sim \boldsymbol{N}\left(\widetilde{\mu}, \widetilde{\sigma}_\mu^2\right),$$

where $\widetilde{\sigma}_\mu^2 = \frac{\sigma^2}{n}$ and $\widetilde{\mu} = \boldsymbol{1}_n^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_0)$.

5. Simulate the full conditional distribution of σ^2:

$$\sigma^2 \mid \mu, \boldsymbol{\beta}_0, \sigma_\beta^2 \sim \chi^{-2}_{\widetilde{\nu},\widetilde{S}},$$

where $\widetilde{\nu} = \nu + n$ and $\widetilde{S} = S + \|\boldsymbol{y} - \boldsymbol{1}_n\mu - \boldsymbol{X}_1\boldsymbol{\beta}_0\|^2$.

6. Repeat steps 2–5 depending on how many values of the parameter vector $(\boldsymbol{\beta}^{\mathrm{T}}, \sigma_\beta^2, \sigma^2)$ you wish to simulate. Usually a large number of iterations are needed and an early part of them are discarded, to finally average the rest of each parameter to obtain estimates of them.

The Gibbs sampler described above can be implemented easily with the BGLR R package: if the hyperparameters S-ν and S_β-ν_β are not specified, by default the BGLR function assigns $\nu = \nu_\beta = 5$, and to S and S_β assigns values such that the mode of the priors of σ^2 and σ_β^2 (inverse scaled Chi-square) matches a certain proportion of the total variance ($1 - R^2$ and R^2): $S = \mathrm{Var}(Y) \times (1 - R^2) \times (\nu + 2)$ and $S_\beta = \mathrm{Var}(Y) \times R^2 \times (\nu_\beta + 2)$ (see Appendix 2 for more details). Explicitly, in BGLR this model can be implemented by running the following R code:

```
ETA = list( list( model = 'BRR', X = X1, df0 = vβ, S0 = Sβ, R2 = 1-R2) )
A = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, S0 = S, df0 = v, R2 = R2)
```

where nIter = 1e4 and burnIn = 1e3 are the desired number of iterations and the number of them to be discarded when computing the estimates of the parameters. Remember that when the hyperparameter values are not given, they are set up in the default values, as previously described.

A sub-model of the BRR that does not induce shrinkage of the beta coefficients is obtained by assuming that $(\beta_1, \ldots, \beta_p)^{\mathrm{T}} \mid \sigma_\beta^2 \sim N_p\left(\boldsymbol{0}, \boldsymbol{I}_p\sigma_\beta^2\right)$, ignoring the prior distribution of σ_β^2 and setting this at a very high value (10^{10}). Note that this model is very similar to the Bayesian model obtained by adopting the prior (6.2), under which the beta coefficients are estimated solely with the information contained in the likelihood function (Pérez and de los Campos 2014). This prior model can also be implemented in the BGLR package and is called **FIXED**. Certainly, the Gibbs

sampler steps for its implementation are the same as the steps described before for the BRR, except that step 2 is removed (no simulations are obtained from σ^2_β) and σ^{-2}_β is set equal to zero in the full conditional of $\boldsymbol{\beta}_0$ (step 3).

6.3 Bayesian GBLUP Genomic Model

In genomic-enabled prediction, the number of markers used to predict the performance of a trait of interest is often very large compared to the number of individuals phenotyped in the sample ($p \gg n$); for this reason, some computational difficulties may arise when exploring the posterior distribution of the beta coefficients. When the main objective is to use this model for predictive purposes, a solution consists of reducing the dimension of the problem by directly simulating values of $\boldsymbol{g} = \boldsymbol{X}_1\boldsymbol{\beta}_0$ (breeding values or genomic effects, Lehermeier et al. 2013) instead of only from $\boldsymbol{\beta}_0$. To do this, first note that because $\boldsymbol{\beta}_0 \mid \sigma^2_\beta \sim N_p\left(\boldsymbol{0}, \boldsymbol{I}_p\sigma^2_\beta\right)$, to induce a prior for $\boldsymbol{g}$, this is defined as $\boldsymbol{g} = \boldsymbol{X}_1\boldsymbol{\beta}_0 \mid \sigma^2_\beta \sim N_n\left(\boldsymbol{0}, \sigma^2_\beta\boldsymbol{X}_1\boldsymbol{X}_1^{\mathrm{T}}\right) = N_n\left(\boldsymbol{0}, \sigma^2_g\boldsymbol{G}\right)$, where $\sigma^2_g = p\sigma^2_\beta$ and $\boldsymbol{G} = \frac{1}{p}\boldsymbol{X}_1\boldsymbol{X}_1^{\mathrm{T}}$, which is known as the genomic relationship matrix (VanRaden 2007). Then, under this parameterization ($\boldsymbol{g} = \boldsymbol{X}_1\boldsymbol{\beta}_0$ and $\sigma^2_g = p\sigma^2_\beta$), the model specified in (6.3), in matrix notation takes the following form:

$$\boldsymbol{Y} = 1_n\mu + \boldsymbol{g} + \epsilon \tag{6.4}$$

with a flat prior to mean parameter (μ), $\sigma^2 \sim \chi^{-2}_{v,S}$, and the induced priors: $\boldsymbol{g} = \boldsymbol{X}_1\boldsymbol{\beta}_0 \mid \sigma^2_g \sim N_n\left(\boldsymbol{0}, \sigma^2_g\boldsymbol{G}\right)$ and $\sigma^2_g \sim \chi^{-2}_{v_g,\ S_g}$ ($v_g = v_\beta$, $S_g = pS_\beta$).

Similarly to what was done for model (6.3), the full conditional posterior distribution of $\boldsymbol{g}$ in model (6.4) is given by

$$\begin{aligned} f(\boldsymbol{g}|-) &\propto L\left(\mu, \boldsymbol{g}, \sigma^2; \boldsymbol{y}\right) f\left(\boldsymbol{g}|\sigma^2_g\right) \\ &\propto \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left[-\frac{1}{2\sigma^2}\|\boldsymbol{y} - 1_n\mu - \boldsymbol{g}\|^2\right] \frac{1}{\left(\sigma^2_g\right)^{\frac{n}{2}}} \exp\left(\left[-\frac{1}{2\sigma^2_g}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{g}\right]\right) \\ &\propto \exp\left\{-\frac{1}{2}\left[(\boldsymbol{g} - \widetilde{\boldsymbol{g}})^{\mathrm{T}}\widetilde{\boldsymbol{G}}^{-1}(\boldsymbol{g} - \widetilde{\boldsymbol{g}})\right]\right\}, \end{aligned}$$

where $\widetilde{\boldsymbol{G}} = \left(\sigma^{-2}_g\boldsymbol{G}^{-1} + \sigma^{-2}\boldsymbol{I}_n\right)^{-1}$ and $\widetilde{\boldsymbol{g}} = \sigma^{-2}\widetilde{\boldsymbol{G}}(\boldsymbol{y} - \boldsymbol{1}_n\mu)$, and from here $\boldsymbol{g} \mid - \sim N_n\left(\widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{G}}\right)$. Then, the mean/mode of $\boldsymbol{g} \mid -$ is $\widetilde{\boldsymbol{g}} = \sigma^{-2}\widetilde{\boldsymbol{G}}(\boldsymbol{y} - \boldsymbol{1}_n\mu)$, which is also the best linear unbiased predictor (BLUP) of $\boldsymbol{g}$ under the mixed model equation of Henderson (1975) using the machinery of a classic linear mixed model described in the previous chapter for model (6.4), after recognizing the

prior distribution of $\boldsymbol{g}$ as the distribution of the random effects in a model, ignoring the priors specification of the rest of the parameters and assuming that they are known (Henderson 1975). For this reason, model (6.4) is often referred to as GBLUP. If $\boldsymbol{G}$ is replaced by the pedigree matrix $\boldsymbol{A}$, the resulting model is known as PBLUP or ABLUP.

The full conditional posterior of the rest of parameters is similar to the BRR model: $\mu \mid - \sim \boldsymbol{N}\left(\widetilde{\mu}, \widetilde{\sigma}_0^2\right)$, where $\widetilde{\sigma}_0^2 = \frac{\sigma^2}{n}$ and $\widetilde{\mu} = \frac{1}{n}\mathbf{1}_n^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{g})$; $\sigma^2 \mid - \sim \chi^{-2}_{\widetilde{\nu},\widetilde{S}}$, where $\widetilde{\nu} = \nu + n$ and $\widetilde{S} = S + \|\boldsymbol{y} - \mathbf{1}_n\mu - \boldsymbol{g}\|^2$; and $\sigma_g^2 \mid - \sim \chi^{-2}_{\widetilde{\nu}_g,\widetilde{S}_g}$, where $\widetilde{\nu}_g = \nu_g + n$ and $\widetilde{S}_g = S_g + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{g}$.

Note that when $p \gg n$, then the dimension of the parameter space of the posterior of GBLUP model is lower than the BRR.

The GBLUP model (6.4) also can be implemented easily with the BGLR R package, and when the hyperparameters S-ν and S_g-ν_g are not specified, $\nu = \nu_g = 5$ is used by default and the scale parameters are settled similarly as in the BRR.

The BGLR code to fit this model:

```
ETA = list( list( model = 'RHKS', K = G, df0 = v_g, S0 = S_g, R2 = 1-R^2)) )
A = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, S0 = S, df0 = v, R2 = R^2)
```

The GBLUP can be equivalently expressed and consequently fitted with the BRR model by making the design matrix equal to the lower triangular factor of the Cholesky decomposition of the genomic relationship matrix, i.e., $\boldsymbol{X} = \boldsymbol{L}$, where $\boldsymbol{G} = \boldsymbol{L}\boldsymbol{L}'$. So, with the BGLR package, the BRR implementation of a GBLUP model is.

```
L = t(chol(G))
ETA = list( list( model = 'BRR', X = L, df0 = v_β, S0 = S_β, R2 = 1-R^2) )
A = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, S0 = S, df0 = v, R2 = R^2)
```

When there is more than one repetition of an individual in the data at hand, or a more sophisticated design is used in the data collection, model (6.4) can be specified in a more general way to take into account this structure, as follows:

$$\boldsymbol{Y} = \mathbf{1}_n\mu + \boldsymbol{Z}\boldsymbol{g} + \boldsymbol{\epsilon} \tag{6.5}$$

with $\boldsymbol{Z}$ the incident matrix of the genotypes. This model cannot be fitted directly in the BGLR and some precalculus is needed first to compute the "covariance" matrix of the predictor $\boldsymbol{Z}\boldsymbol{g}$ in model (6.5): $\mathbf{K}_L = \mathrm{Var}(\boldsymbol{Z}\boldsymbol{g}) = \boldsymbol{Z}\boldsymbol{G}\boldsymbol{Z}^{\mathrm{T}}$. The BGLR code for implementing this model is the following:

```
Z = model.matrix(~0+GID,data=dat_F,xlev = list(GID=unique
(dat_F$GID)))
K_L = Z%*%G%*%t(Z)
ETA = list( list( model = 'RHKS', K = K_L, df0 = v_g, S0 = S_g, R2 = 1-R^2)) )
A = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, S0 = S, df0 = v, R2 = R^2)
```

where **dat_F** is the data set that contains the necessary phenotypic information (GID: Lines or individuals; y: response variable of the trait of interest).

6.4 Genomic-Enabled Prediction BayesA Model

Another variant to the standard Bayesian model (6.1) is the BayesA model proposed by Meuwissen et al. (2001), which is a slight modification of the BRR model obtained with the same prior distributions, except that now a specific variance $\sigma^2_{\beta_j}$ is assumed for each of the covariate (marker) effects, that is, $\beta_j \mid \sigma^2_{\beta_j} \sim N\left(0, \sigma^2_{\beta j}\right)$, and these variance parameters are supposed to be independent random variables with a scaled inverse Chi-square distribution with parameters v_β and S_β, $\sigma^2_{\beta_j} \sim \chi^{-2}\left(v_\beta, S_\beta\right)$. These specific variances for each marker effect provide covariate heterogeneous shrinkage estimation. Furthermore, a gamma distribution is assigned to S_β, $S_\beta \sim G(r, s)$, where $G(r, s)$ denotes a gamma distribution where r and s are the rate and shape parameters, respectively. By providing a different prior variance for each β_j, this model has the potential of inducing covariate-specific shrinkage of estimated effects (Pérez and de los Campos 2013).

Note that choosing $r = r^*/v_\beta$ and taking very large values of v_β, the prior of $\sigma^2_{\beta j}$ collapses to a degenerate distribution at S_β, and the BRR model is obtained, but with a gamma distribution with parameters r^* and s as priors to the common variance of the effects $\sigma^2_\beta = \mathrm{Var}\left(\beta_j\right) = S_\beta$, instead of $\chi^{-2}(v_\beta, S_\beta)$. Furthermore, the marginal distribution of each beta coefficient β_j, that is, the unconditional distribution of $\beta_j \mid S_\beta$, is a scaled-t-student distribution (scaled by $\sqrt{S_\beta / v_\beta}$). These distributions, compared to the normal, have heavier tails and put higher mass around 0, which compared to the BRR, induce fewer shrinkage estimates of covariates with sizable effects, and induce strong shrinkage toward zero estimates of covariates with smaller effects, respectively (de los Campos et al. 2013).

A Gibbs sampler implementation for estimating the parameters of this model can be done following steps 1–6 of the BRR model, where the second step is replaced by the next 2.1 and 2.2 steps, the third and the last step are replaced and modified by the next steps 3 and 6:

2.1 Simulate from the full conditional of each $\sigma^2_{\beta_j}$

$$\sigma^2_{\beta_j} \mid \mu, \boldsymbol{\beta}_0, \boldsymbol{\sigma}^2_{-j}, S_\beta, \sigma^2 \sim \chi^{-2}_{\widetilde{v}_j, \widetilde{S}_{\beta_j}},$$

where $\widetilde{v}_{\beta_j} = v_\beta + 1$ and scale parameter $\widetilde{S}_{\beta_j} = S_\beta + \beta^2_j$, where $\boldsymbol{\sigma}^2_{-j}$ is the vector $\boldsymbol{\sigma}^2_\beta = \left(\sigma^2_{\beta_1}, \ldots, \sigma^2_{\beta_p}\right)$ but without the jth entry.

2.2 Simulate from the full conditional of S_β

$$
\begin{aligned}
f(S_\beta|-) &\propto \left[\prod_{j=1}^{p} f\left(\sigma^2_{\beta_j}|S_\beta\right)\right] f(S_\beta) \\
&\propto \prod_{j=1}^{p}\left[\frac{\left(\frac{S_\beta}{2}\right)^{\frac{v_\beta}{2}}}{\Gamma\left(\frac{v_\beta}{2}\right)\left(\sigma^2_{\beta_j}\right)^{1+\frac{v_\beta}{2}}}\exp\left(-\frac{S_\beta}{2\sigma^2_{\beta_j}}\right)\right] S_\beta^{s-1}\exp\left(-rS_\beta\right) \\
&\propto S_\beta^{s+\frac{pv_\beta}{2}-1}\exp\left[-\left(r+\frac{1}{2}\sum_{j=1}^{p}\frac{1}{\sigma^2_{\beta j}}\right)S_\beta\right]
\end{aligned}
$$

which corresponds to the kernel of the gamma distribution with rate parameter $\widetilde{r} = r + \frac{1}{2}\sum_{j=1}^{p}\frac{1}{\sigma^2_{\beta j}}$ and shape parameter $\widetilde{s} = s + \frac{pv_\beta}{2}$, and so $S_\beta \mid - \sim \text{Gamma}(\widetilde{r}, \widetilde{s})$.

3. Simulate the full conditional posterior distribution of $\boldsymbol{\beta}_0$:

$$\boldsymbol{\beta}_0 \mid \mu, \boldsymbol{\sigma}^2_\beta, \sigma^2 \sim N_p\left(\widetilde{\boldsymbol{\beta}}_0, \widetilde{\boldsymbol{\Sigma}}_0\right),$$

where $\widetilde{\boldsymbol{\Sigma}}_0 = \left(\boldsymbol{D}_p^{-1} + \sigma^{-2}\boldsymbol{X}_1^T\boldsymbol{X}_1\right)^{-1}$ and $\widetilde{\boldsymbol{\beta}}_0 = \sigma^{-2}\widetilde{\boldsymbol{\Sigma}}_0\boldsymbol{X}_1^{\mathbf{T}}(\boldsymbol{y} - \mathbf{1}_n\mu)$, $\boldsymbol{D}_p = \text{Diag}\left(\sigma^2_{\beta_1}, \ldots, \sigma^2_{\beta_p}\right)$.

6. Repeat steps 2–5 (given in the BRR method) depending on how many values of the parameter vector $(\boldsymbol{\beta}^{\mathrm{T}}, \boldsymbol{\sigma}^2_\beta, \sigma^2, S_\beta)$ we wish to simulate.

When implementing this model in the BGLR package, by default $v = v_\beta = 5$ are used and $S = \text{Var}(Y) \times (1 - R^2) \times (v + 2)$, which makes the mode of the priors of σ^2 $(\chi^{-2}(v, S))$ match a certain proportion of the total variance $(1 - R^2)$. If the hyperparameters of the a priori for S_β, s, and r, are not specified, by default BGLR takes $s = 1.1$ to have an a priori $(S_\beta \sim G(s, r))$ that is relatively non-informative with a coefficient of variation $(1/\sqrt{s})$ of approximately 95%. Then, to the rate parameter it assigns the value $r = (s - 1)/S_\beta$, with $S_\beta = \text{Var}(y) \times R^2 \times \left(v_\beta + 2\right)/S_x^2$, where S_x^2 is the sum of the variances of the columns of $\boldsymbol{X}$ and R^2 is the percentage of the total variation that a priori is required to attribute to the covariates in $\boldsymbol{X}$. The BGLR code for implementing this model is

```
ETA = list( list( model = 'BayesA', X=X_1, df0 = v_β, rate0 = r, shape0 = s,
R2 = 1-R^2))
A   = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, S0 = S, df0 = v, R2 =
R^2)
```

6.5 Genomic-Enabled Prediction BayesB and BayesC Models

Other variants of the model (6.1) are the BayesC and the BayesB models, which in turn can be considered as direct extensions of the BRR and BayesA models, respectively, by adding a parameter π that represents the prior proportion of covariates with nonzero effect (Pérez and de los Campos 2014).

The **BayesC** is the same as the BRR, but instead of assuming a priori that the beta coefficients are independent normal random variables with mean 0 and variance σ^2_β, it assumes that with probability π each β_j comes from a $N\left(0,\sigma^2_\beta\right)$, and with probability $1-\pi$ comes from a degenerate distribution (DG) at zero, that is, $\beta_1,\ldots,\beta_p \mid \sigma^2_\beta,\pi \overset{iid}{\sim} \pi_p \mathrm{N}\left(0,\sigma^2_\beta\right)+\left(1-\pi_p\right)\mathrm{DG}(0)$ (mix of a normal distribution with mean 0 and variance σ^2_β, and degenerate distribution at zero). In addition, for the parameter π_p, a beta distribution is assigned as prior, that is, π_p~Beta (π_{p0}, ϕ_0), where $\pi_{p0}=E(\pi_p)$ represents the mean and ϕ_0^{-1} is the "dispersion" parameter ($\mathrm{var}\left(\pi_p\right)=\frac{\pi_{p0}\left(1-\pi_{p0}\right)}{\phi_0+1}$). If $\phi_0=2$ and $\pi_{p0}=0.5$, the prior for π_p is a uniform distribution in (0,1). For large values of ϕ_0, the distribution for π_p is highly concentrated around π_{p0}, and so the BayesC is reduced to BRR when $\pi_{p0}=1$ for large values of ϕ_0.

For this model, the full conditional distributions of μ and σ^2_β are the same as the model described before, that is, $\mu \mid -\sim N\left(\widetilde{\mu},\widetilde{\sigma}^2_\mu\right)$ and $\sigma^2 \mid -\sim \underset{\sim}{\chi}^{-2}_{\tilde{\nu},\tilde{S}}$. However, for the rest of the parameters, this does not have a known form and is not easy to simulate from them. A solution is to introduce a latent variable to represent the prior distribution of each β_j, and compute all the conditional distributions in this augmented scheme, including the distribution corresponding to the latent variable. To do this, note that this prior can be specified by assuming that conditional to a binary latent variable Z_j,

$$\beta_j \mid \sigma^2_\beta, Z_j=z \sim \begin{cases} N\left(0,\sigma^2_\beta\right), \\ \mathrm{DG}(0), \end{cases}$$

where Z_j is a Bernoulli random variable with parameter π_p ($Z_j \sim \mathrm{Ber}(\pi_p)$). With this introduced latent variable, all the full conditionals can be derived, as is described next.

If the current value of z_j is 1, the full conditional posterior of β_j is

$$
\begin{aligned}
f(\beta_j|-) &\propto L(\mu,\boldsymbol{\beta}_0,\sigma^2;\boldsymbol{y})f\left(\beta_j|\sigma^2_\beta,z_j\right)\\
&\propto \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{y}-1_n\mu-\boldsymbol{X}_1\boldsymbol{\beta}_0\|\right)\frac{1}{\sqrt{2\pi\sigma^2_\beta}}\exp\left(-\frac{\beta_j^2}{2\sigma^2_\beta}\right)\\
&\propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_{ij}-x_{ij}\beta_j\right)^2-\frac{1}{2\sigma^2_\beta}\beta_j^2\right)\\
&\propto \exp\left\{-\frac{1}{2}\left[\left(\sigma_\beta^{-2}+\sigma^{-2}\sum_{i=1}^{n}x_{ij}^2\right)\beta_j^2-2\sigma^{-2}\sum_{i=1}^{n}x_{ij}y_{ij}\beta_j+\frac{1}{\sigma^2}\sum_{i=1}^{n}y_{ij}^2\right]\right\}\\
&\propto \exp\left[-\frac{\left(\beta_j-\widetilde{\beta}_j\right)^2}{2\widetilde{\sigma}_j^2}\right],
\end{aligned}
$$

where $y_{ij}=y_i-\sum_{k=1,\,k\neq j}^{p}x_{ik}\beta_k$, $\widetilde{\sigma}_j^2=\left(\sigma_\beta^{-2}+\sigma^{-2}\sum_{i=1}^{n}x_{ij}^2\right)^{-1}$, and $\widetilde{\beta}_j=\sigma^{-2}\widetilde{\sigma}_j^2\sum_{i=1}^{n}x_{ij}y_{ij}$. That is, when the current value of z_j is 1, $\beta_j\mid -\sim N\left(\widetilde{\beta}_j,\widetilde{\sigma}_j^2\right)$. However, if $z_j=0$, the full conditional posterior of β_j is a degenerate random variable at 0, that is, $\beta_j\mid -\sim \mathrm{DG}(0)$.

The full conditional distribution of Z_j is

$$
\begin{aligned}
f(z_j|-) &\propto f\left(\beta_j|\sigma^2_\beta,z_j\right)f(z_j)\\
&\propto f\left(\beta_j|\sigma^2_\beta,z_j\right)\pi_p^{z_j}\left(1-\pi_j\right)^{1-z_j}
\end{aligned}
$$

from which, conditional on the rest of the parameters, Z_j is a Bernoulli random variable with parameter $\widetilde{\pi}_{pj}=\dfrac{\frac{\pi_p}{\sqrt{2\pi\sigma^2_\beta}}\exp\left(-\frac{\beta_j^2}{2\sigma^2_\beta}\right)}{\frac{\pi_p}{\sqrt{2\pi\sigma^2_\beta}}\exp\left(-\frac{\beta_j^2}{2\sigma^2_\beta}\right)+\left(1-\pi_p\right)\delta_0\left(\beta_j\right)}$. Note however that, if $\beta_j\neq 0$, $\widetilde{\pi}_{pj}=1$, and then $Z_j=1$ with probability 1, when simulating from the full conditional posterior of β_j, we will always obtain values different from zero, and this cyclic behavior will remain permanent. On the other hand, note that if $\beta_j=0$, $\widetilde{\pi}_{pj}=\dfrac{\frac{\pi_p}{\sqrt{2\pi\sigma^2_\beta}}}{\frac{\pi_p}{\sqrt{2\pi\sigma^2_\beta}}+\left(1-\pi_p\right)}$ is not 0, then the next simulated value of β_j will be different from 0, and in this scenario, in the next steps of the "Gibbs," Z_j will at all times be 1, and so the chain has absorbing states and will not explore the entire sampling space. A solution to this problem consists of trying to simulate from the joint conditional distribution of β_j and Z_j, that is, from $\beta_j, Z_j\mid -$. This full joint conditional distribution can be computed as

$$f(\beta_j, z_j|-) \propto f_*(z_j) f(\beta_j|z_j, -) \propto f_*(z_j) L(\mu, \boldsymbol{\beta}_0, \sigma^2; \boldsymbol{y}) f\left(\beta_j|\sigma^2_\beta, z_j\right),$$

where $f_*(z_j)$ is the marginal conditional distribution of Z_j conditioned to all parameters except β_j ($Z_j \mid -j \sim f_*(\cdot)$). Specifically, this is given by "integrating" ($f(\beta_j, z_j|-)$ with respect to β_j)

$$\begin{aligned}
f_*(z_j) &\propto \int_{-\infty}^{\infty} f(\beta_j, z_j|-) d\beta_j \propto \int_{-\infty}^{\infty} L(\mu, \boldsymbol{\beta}_0, \sigma^2; \boldsymbol{y}) f\left(\beta_j|\sigma^2_\beta, z_j\right) f(z_j|\pi_p) d\beta_j \\
&\propto \begin{cases} \int_{-\infty}^{\infty} \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_{ij} - x_{ij}\beta_j\right)^2\right] \frac{1}{\sqrt{2\pi\sigma^2_\beta}} \exp\left(-\frac{\beta_j^2}{2\sigma^2_\beta}\right) \pi_p d\beta_j \\ \quad \int_{-\infty}^{\infty} \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_{ij} - x_{ij}\beta_j\right)^2\right] \delta_0(\beta_j)(1-\pi_p) d\beta_j \end{cases} \\
&\propto \begin{cases} \pi_p \sqrt{\frac{\widetilde{\sigma}^2_j}{\sigma^2_\beta}} \\ 1 - \pi_p. \end{cases}
\end{aligned}$$

From here, $Z_j \mid -$ is a Bernoulli random distribution with parameter $\widetilde{\pi}_p = \left(\pi_p \sqrt{\frac{\widetilde{\sigma}^2_j}{\sigma^2_\beta}}\right) / \left(\pi_p \sqrt{\frac{\widetilde{\sigma}^2_j}{\sigma^2_\beta}} + 1 - \pi_p\right)$. With this and the full conditional distribution derived above for β_j, an easy way to simulate values from β_j, $Z_j \mid -$ consists of first simulating a z_j value from $Z_j \mid -j \sim \text{Ber}(\widetilde{\pi}_p)$, and then, if $z_j = 1$, simulating a value of β_j from $\beta_j \mid - \sim N\left(\widetilde{\beta}_j, \widetilde{\sigma}^2_j\right)$, otherwise take $\beta_j = 0$.

Now, note that the full conditional distribution of σ^2_β is

$$\begin{aligned}
f\left(\sigma^2_\beta|-\right) &\propto \left\{\prod_j^p \left[f\left(\beta_j|\sigma^2_\beta, z_j\right)\right]^{z_j}\right\} f\left(\sigma^2_\beta\right) \\
&\propto \frac{1}{\left(\sigma^2_\beta\right)^{p\bar{z}_p}} \exp\left(-\frac{1}{2\sigma^2_\beta}\sum_{j=1}^{p} z_j \beta_j^2\right) \frac{\left(\frac{S_\beta}{2}\right)^{\frac{v_\beta}{2}}}{\Gamma\left(\frac{v_\beta}{2}\right)\left(\sigma^2_\beta\right)^{1+\frac{v_\beta}{2}}} \exp\left(-\frac{S_\beta}{2\sigma^2_\beta}\right) \\
&\propto \frac{1}{\left(\sigma^2_\beta\right)^{p\bar{z}_p}} \exp\left(-\frac{1}{2\sigma^2_\beta}\sum_{j=1}^{p} z_j \beta_j^2\right) \frac{\left(\frac{S_\beta}{2}\right)^{\frac{v_\beta}{2}}}{\Gamma\left(\frac{v_\beta}{2}\right)\left(\sigma^2_\beta\right)^{1+\frac{v_\beta}{2}}} \exp\left(-\frac{S_\beta}{2\sigma^2_\beta}\right) \\
&\propto \frac{1}{\left(\sigma^2_\beta\right)^{1+\frac{\widetilde{v}_\beta}{2}}} \exp\left(-\frac{\widetilde{S}_\beta}{2\sigma^2_\beta}\right),
\end{aligned}$$

where $\bar{z}_p = 1/p\sum_{j=1}^{p} z_j$, $\widetilde{S}_\beta = S_\beta + \sum_{j=1}^{p} z_j \beta_j^2$, and $\widetilde{\nu}_\beta = \nu_\beta + p\bar{z}_p$. That is, $\sigma_\beta^2 \mid \mu, \boldsymbol{\beta}_0, \sigma^2, \mathbf{z} \sim \chi^{-2}_{\widetilde{\nu},S_\beta}$. The full conditional distributions of μ and σ^2 are the same as BRR, that is, $\mu \mid - \sim N\left(\widetilde{\mu}, \widetilde{\sigma}_\mu^2\right)$ and $\sigma^2 \mid - \sim \chi^{-2}_{\widetilde{\nu},S}$, with $\widetilde{\sigma}_\mu^2 = \frac{\sigma^2}{n}$, $\widetilde{\mu} = \mathbf{1}_n^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_0)$, $\widetilde{\nu} = \nu + n$, and $\widetilde{S} = S + \|\boldsymbol{y} - \mathbf{1}_n\mu - \boldsymbol{X}_1\boldsymbol{\beta}_0\|^2$.

The full conditional distribution of π_p is

$$
\begin{aligned}
f(\pi_p|-) &\propto \left[\prod_{j=1}^{p} f(z_j|\pi_p)\right] f(\pi_p) \\
&\propto \pi_p^{p\bar{z}_p}\left(1-\pi_p\right)^{p\left(1-\bar{z}_p\right)} \pi_p^{\phi_0\pi_{p0}+1-1}\left(1-\pi_p\right)^{\phi_0\left(1-\pi_{p0}\right)-1} \\
&\propto \pi_p^{\phi_0\pi_{p0}+p\bar{z}_p-1}\left(1-\pi_p\right)^{\phi_0\left(1-\pi_{p0}\right)+p\left(1-\bar{z}_p\right)-1}
\end{aligned}
$$

which means that $\pi_p \mid - \sim \text{Beta}\left(\widetilde{\pi}_{p0}, \widetilde{\phi}_0\right)$, with $\widetilde{\phi}_0 = \phi_0 + p$ and $\widetilde{\pi}_{p0} = \frac{\phi_0\pi_{p0}+p\bar{z}_p}{\phi_0+p}$.

The **BayesB** model is a variant of BayesA that assumes almost the same prior models to the parameters, except that instead of assuming independent normal random variables with common mean 0 and common variance σ_β^2 for the beta coefficients, this model adopts a mixture distribution, that is, $\beta_j \mid \sigma^2_{\beta_j}, \pi \overset{iid}{\sim} \pi N\left(0, \sigma^2_{\beta_j}\right) + (1-\pi)\text{DG}(0)$, with $\pi \sim \text{Beta}(\pi_0, p_0)$. This model has the potential to perform variable selection and produce covariate-specific shrinkage estimates (Pérez et al. 2010).

This model can also be considered an extension of the BayesC model with a gamma distribution as prior to the scale parameter of the a priori distribution of the variance of the beta coefficients, that is, $S_\beta \sim G(s, r)$. It is interesting to point out that if $\pi = 1$, this model is reduced to BayesA, which is obtained by taking $\pi_0 = 1$ and letting ϕ_0 go to ∞. Also, this is reduced to the BayesC by setting $s/r = S_\beta^0$ and choosing a very large value for r.

To explore the posterior distribution of this model, the same Gibbs sampler given for BayesC can be used, but adding to the process the full conditional posterior distribution of S_β: $S_\beta \mid - \sim \text{Gamma}(\widetilde{r}, \widetilde{s})$, where $\widetilde{r} = r + \frac{1}{2\sigma_\beta^2}$ and shape parameter $\widetilde{s} = s + \frac{\nu_\beta}{2}$.

When implementing both these models in the BGLR R package, by default this assigns $\pi_{p0} = 0.5$ and $\phi_0 = 10$, for the hyperparameters of the prior model of π_p, Beta (π_{p0}, ϕ_0), which results in a weakly informative prior. For the remaining hyperparameters of the **BayesC** model, by default BGLR assigns values like those assigned to the BRR model, but with some modifications to consider because a priori now only a proportion π_0 of the covariates (columns of $\boldsymbol{X}$) has nonzero effects:

$$v = v_\beta = 5,$$
$$S = \text{Var}(Y) \times \left(1 - R^2\right) \times (v + 2),$$
$$S_\beta = \text{Var}(y) \times R^2 \times \frac{\left(v_\beta + 2\right)}{S_x^2 \pi_0}.$$

While for the remaining hyperparameters of **BayesB**, by default BGLR also assigns values similar to **BayesA**: $v = v_\beta = 5$, $S = \text{Var}(Y) \times (1 - R^2) \times (v + 2)$, $r = (s - 1)/S_\beta$, with $S_\beta = \text{Var}(y) \times R^2 \times \frac{(v_\beta + 2)}{S_x^2 \pi_0}$, where S_x^2 is the sum of the variances of the columns of $\boldsymbol{X}$.

The BGLR codes to implement these models are, respectively:

```
ETA = list( list( model = 'BayesC', X=X1 ) )
A   = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, df0 = v, S0 = S, probIn
= π0, counts = φ0, R2 = R2)
```

and

```
ETA = list( list( model = 'BayesB', X=X1 ) )
A   = BGLR(y=y, ETA = ETA, nIter = 1e4, burnIn = 1e3, df0 = v, rate0 = r,
shape0 = s, probIn = π0, counts = φ0, R2 = R2)
```

6.6 Genomic-Enabled Prediction Bayesian Lasso Model

Another variant of the model (6.1) is the Bayesian Lasso linear regression model (**BL**). This model assumes independent Laplace or double-exponential distributions with location and scale parameters 0 and $\frac{\sqrt{\sigma^2}}{\lambda}$, respectively, for the beta coefficients, that is, $\beta_1, \ldots, \beta_p \mid \sigma^2, \lambda \overset{iid}{\sim} L\left(0, \frac{\sqrt{\sigma^2}}{\lambda}\right)$. Furthermore, the priors for parameters μ and σ^2 are the same as in the models described before, while for λ^2, a gamma distribution with parameters s_λ and r_λ is often adopted.

Because compared to the normal distribution, the Laplace distribution has fatter tails and puts higher density around 0, this prior induces stronger shrinkage estimates for covariates with relatively small effects and reduced shrinkage estimates for covariates with larger effects (Pérez et al. 2010).

A more convenient specification of the prior for the beta coefficients in this model is obtained with the representation proposed by Park and Casella (2008), which is a continuous scale mixture of a normal distribution: $\beta_j \mid \tau_j \sim N(0, \tau_j \sigma^2)$ and $\tau_j \sim \text{Exp}(2/\lambda^2)$, $j = 1, \ldots, p$, where Exp(θ) denotes an exponential distribution with scale parameter θ. So, unlike the prior used by the BRR model, this prior distribution also puts a higher mass at zero and has heavier tails, which induce stronger shrinkage estimates for covariates with relatively small effect and less shrinkage estimates for markers with sizable effect (Pérez et al. 2010).

Note that the prior distribution for the beta coefficients and the prior variance of this distribution in BayesB and BayesC can be equivalently expressed as a mixture of a scaled inverse Chi-squared distribution with parameters v_β and S_β, and a degenerate distribution at zero, that is, $\beta_j \sim N\left(0, \sigma^2_\beta\right)$ and $\sigma^2_\beta \sim \pi_p \chi^{-2}\left(v_\beta, S_\beta\right) + \left(1 - \pi_p\right)\mathrm{DG}(0)$. So, based on this result and the connections between the models described before, the main difference between all these models is the manner in which the prior variance of the predictor variable is modelled.

Example 1 To illustrate how to use the models described before, here we consider the prediction of grain yield (tons/ha) based on marker information. The data set used consists of 30 lines in four environments with one and two repetitions and the genotyped information contains 500 markers for each line. The numbers of lines with one (two) repetition are 6 (24), 2 (28), 0 (30), and 3 (27) in Environments 1, 2, 3, and 4, respectively, resulting in 229 observations. The performance prediction of all these models was evaluated with 10 random partitions in a cross-validation strategy, where 80% of the complete data set was used to fit the model and the rest to evaluate the model in terms of the mean squared error of prediction (MSE).

The results for all models (shown in Table 6.1) were obtained by iterating 10,000 times the corresponding Gibbs sampler and discarding the first 1000 of them, using the default hyperparameter values implemented in BGLR. This indicates that the behavior of all the models is similar, except the BayesC, where the MSE is slightly greater than the rest.

The R code to obtain the results in Table 6.1 is given in Appendix 3.

What happens when using other hyperparameter values? Although the ones used here (proposed by Pérez et al. 2010) did not always produce the best prediction performance (Lehermeier et al. 2013) and there are other ways to propose the hyperparameter values in these models (Habier et al. 2010, 2011), it is important to point out that the values used by default in BGLR work reasonably well and that it

Table 6.1 Mean squared error (MSE) of prediction across 10 random partitions, with 80% for training and the rest for testing, in five Bayesian linear models

PT	BRR	GBLUP	BayesA	BayesB	BayesC
1	0.5395	0.5391	0.5399	0.5415	0.5470
2	0.3351	0.3345	0.3345	0.3411	0.3511
3	0.4044	0.4065	0.4056	0.4050	0.4106
4	0.3540	0.3583	0.3571	0.3561	0.3559
5	0.3564	0.3555	0.3556	0.3549	0.3604
6	0.7781	0.7772	0.7702	0.7777	0.7776
7	0.7430	0.7357	0.7380	0.7384	0.7431
8	0.3662	0.3717	0.3695	0.3669	0.3661
9	0.3065	0.3026	0.3056	0.3021	0.3030
10	0.5842	0.5822	0.5846	0.5860	0.6079
Mean (SD)	0.4767 (0.1742)	0.4763 (0.1724)	0.476 (0.1716)	0.4769 (0.1734)	0.4822 (0.1744)

is not easy to find other combinations that work better in all applications, and when you want to use other combinations of hyperparameters you need to be very careful because you can dramatically affect the predictive performance of the model that uses the default hyperparameters.

Indeed, by means of simulated and experimental data, Lehermeier et al. (2013) observed a strong influence on the predictive performance of the hyperparameters given to the prior distributions in BayesA, BayesB, and the Bayes Lasso with fixed λ. Specifically, in the first two models, they observed that the scale parameter S_β of the prior distribution of variance of β_j had a strong effect on the predictive ability because overfitting in the data occurred when a too large value of this value was chosen, whereas underfitting was observed when too small values of this parameter were used. Note that this is expected approximately by seeing that in both models (BayesA and BayesB), $\text{Var}(\beta_j) = E\left(\sigma^2_{\beta_j}\right) = S_\beta/(\nu_\beta - 1)$, which is almost the inverse of the regularization parameter in any type of Ridge regression model.

6.7 Extended Predictor in Bayesian Genomic Regression Models

All the Bayesian formulations of the model (6.1) described before can be extended, in terms of the predictor, to easily take into account the effects of other factors. For example, effects of environments and environment–marker interaction can be added as

$$\boldsymbol{y} = \mathbf{1}_n\mu + \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{X}_{EM}\boldsymbol{\beta}_{EM} + \boldsymbol{\epsilon}, \tag{6.6}$$

where $\boldsymbol{X}_E$ and $\boldsymbol{X}_{EM}$ are the design matrices of the environments and environment–marker interactions, respectively, while $\boldsymbol{\beta}_E$ and $\boldsymbol{\beta}_{EM}$ are the vectors of the environment effects and the interaction effects, respectively, with a prior distribution that can be specified as was done for $\boldsymbol{\beta}$. Indeed, with the BGLR function all these things are possible, and all the options described before can also be adopted for the rest of effects added in the model: FIXED, BRR, BayesA, BayesB, BayesC, and BL.

Under the RKHS model with genotypic and environment–genotypic interaction effects, in the predictor, the modified model (6.6) is expressed as

$$\boldsymbol{Y} = \mathbf{1}_n\mu + \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{Z}_L\boldsymbol{g} + \boldsymbol{Z}_{EL}\boldsymbol{gE} + \boldsymbol{\epsilon}, \tag{6.7}$$

where $\boldsymbol{Z}_L$ and $\boldsymbol{Z}_{LE}$ are the incident matrices of the genomic and environment–genotypic interaction effects, respectively. Similarly to model (6.5), this model cannot be fitted directly in the BGLR and some precalculations are needed first to compute the "covariance" matrix of the predictors $\boldsymbol{Z}_L\boldsymbol{g}$ and $\boldsymbol{Z}_{EL}\boldsymbol{gE}$, which are $\boldsymbol{K}_L = \sigma_g^{-2}\,\text{Var}(\boldsymbol{Z}_L\boldsymbol{g}) = \boldsymbol{Z}_L\boldsymbol{G}\boldsymbol{Z}_L^{\text{T}}$ and $\boldsymbol{K}_{LE} = \sigma_{gE}^{-2}\text{Var}(\boldsymbol{Z}_{LE}\boldsymbol{gE}) = \boldsymbol{Z}_{LE}(\boldsymbol{I}_I \otimes \boldsymbol{G})\boldsymbol{Z}_{LE}^{\text{T}}$,

respectively, where I is the number of environments. The BGLR code for implementing this model is the following:

```
   I = length(unique(dat_F$Env))
  XE = model.matrix(~0+Env,data=dat_F)[,-1]
  Z_L = model.matrix(~0+GID,data=dat_F,xlev = list(GID=unique
(dat_F$GID)))
  K_L = Z_L %*%G%*%t(Z_L)
  Z_LE = model.matrix(~0+GID:Env,data=dat_F,
         xlev = list(GID=unique(dat_F$GID),Env = unique(dat_F$Env)))
  K_LE = Z_LE%*%kronecker(diag(I),G)%*%t(Z_LE)
       ETA = list( list(model='FIXED',X=XE),
                   list( model = 'RHKS', K = K_L, df0 = v_g, S0 = S_g, R2 = 1-R^2)),
                     list(model='RKHS',K=K_LE )
 A   = BGLR(y,ETA = ETA, nIter = 1e4, burnIn = 1e3, S0 = S, df0 = v, R2 = R^2)
```

where **dat_F** is the data set that contains the necessary phenotypic information (GID: Lines or individuals; Env: Environment; y: response variable of trait under study).

Example 2 (Includes Models with Only Env Effects and Models with Env and LinexEnv Effects) To illustrate how to fit the extended genomic regression models described before, here we consider the prediction of grain yield (tons/ha) based on marker information and the genomic relationship derived from it. The data set used consists of 30 lines in four environments, and the genotyped information of 500 markers for each line. The performance prediction of 18 models was evaluated with a five-fold cross-validation, where 80% of the complete data set was used to fit the model and the rest to evaluate the model in terms of the mean squared error of prediction (MSE). These models were obtained by considering different predictors (marker, environment, or/and environment–marker interaction) and different prior models to the parameters of each predictor included.

The model M1 only considered in the predictor the marker effects, from which six sub-models were obtained by adopting one of the six options (BRR, RKHS, BayesA, BayesB, BayesC, or BL) to the prior model of $\boldsymbol{\beta}$ (or $\boldsymbol{g}$). Model M2 is model M1 plus the environment effects with a FIXED prior model, for all prior sub-models in the marker predictor. Model M3 is model M2 plus the environment–marker interaction, with a prior model of the same family as those chosen for the marker predictor (see in Table 6.2 all the models we compared).

The performance prediction of the models presented in Table 6.2 is shown in Table 6.3. The first column represents the kind of prior model used in both marker effects and env:marker interaction terms, when the latter is included in the model. In each of the first five prior models, model M2 resulted in better MSE performance, while when the BL prior model was used, model M3, the model with the interaction term, was better. The greater difference is between M1 and M2, where the average MSE across all priors of the first model is approximately 21% greater than the corresponding average of the M2 model. Similar behavior was observed with Pearson's correlation, with the average of this criterion across all priors about 32%

Table 6.2 Fitted models: M_{md}, $m = 1, 2, 3$, $d = 1, \ldots, 6$

Model	Sub-model	Predictor		
		Marker or G: $\boldsymbol{X\beta}$	Environment: $\boldsymbol{X_E\beta_E}$	Interaction Env-Markers: $\boldsymbol{X_{EM}\beta_{EM}}$
M1	M11	BRR		
	M12	RKHS		
	M12	BayesA		
	M14	BayesB		
	M15	BayesC		
	M16	BL		
M2	M21	BRR	FIXED	
	M22	RKHS	FIXED	
	M23	BayesA	FIXED	
	M24	BayesB	FIXED	
	M25	BayesC	FIXED	
	M26	BL	FIXED	
M3	M31	BRR	FIXED	BRR
	M32	RKHS	FIXED	RKHS
	M33	BayesA	FIXED	BayesA
	M34	BayesB	FIXED	BayesB
	M35	BayesC	FIXED	BayesC
	M36	BL	FIXED	BL

The third to fifth columns are the considered predictors corresponding to marker (genetic effect), environment, and environment–marker (genetic effect) effects, respectively. The cells in row 2 and column 3 indicate the prior distribution model specified for the beta parameters in each predictor when that parameter is present. M1 : $\boldsymbol{Y = 1\mu + X\beta + \epsilon}$, M2 : $\boldsymbol{Y} = \boldsymbol{1}\mu + \boldsymbol{X_E\beta_E} + \boldsymbol{X\beta} + \boldsymbol{\epsilon}$, and M3 : $\boldsymbol{Y} = \boldsymbol{1}\mu + \boldsymbol{X_E\beta_E} + \boldsymbol{X\beta} + \boldsymbol{X_{EM}\beta_{EM}} + \boldsymbol{\epsilon}$. For the RKHS prior model, predictors $\boldsymbol{X\beta}$ and $\boldsymbol{X_{EM}\beta_{EM}}$ are replaced by $\boldsymbol{g}$ and $\boldsymbol{Eg}$, respectively

greater in model M2 than in M1. So the inclusion of the environment effect was important, but not the environment:marker interaction.

The best prediction performance in terms of MSE was obtained with sub-model M25 (M2 with a BayesC prior) followed by M21 (M2 with a BRR prior). However, the difference between those and sub-models M22, M23, and M24, also derived from M2, is only slight and a little more than with M26, which as commented before, among the models that assume a BL prior, showed a worse performance than M36 (M3 plus a BL prior for marker effects and environment–marker interaction).

6.8 Bayesian Genomic Multi-trait Linear Regression Model

The univariate models described for continuous outcomes do not exploit the possible correlation between traits, when the selection of better individuals is based on several traits and these univariate models are fitted separately to each trait. Relative

Table 6.3 Performance prediction of models in Table 6.2: Mean squared error of prediction (MSE) and average Pearson's correlation (PC), each with its standard deviation across the five partitions

	Markers		Env + Markers		Env + Markers + Env:Markers	
	M1		M2		M3	
Prior	MSE (SD)	PC (SD)	MSE (SD)	PC (SD)	MSE (SD)	PC (SD)
BRR	0.5384 (0.08)	0.4678 (0.11)	0.4446 (0.11)	0.6112 (0.1)	0.4782 (0.12)	0.579 (0.11)
RKHS	0.5509 (0.09)	0.4505 (0.13)	0.4542 (0.12)	0.6012 (0.1)	0.51 (0.13)	0.5271 (0.12)
BayesA	0.5441 (0.08)	0.4622 (0.12)	0.4452 (0.11)	0.616 (0.1)	0.4806 (0.15)	0.5964 (0.11)
BayesB	0.5462 (0.08)	0.4595 (0.12)	0.4455 (0.11)	0.6141 (0.1)	0.4669 (0.13)	0.5981 (0.11)
BayesC	0.5449 (0.08)	0.4615 (0.12)	0.4416 (0.11)	0.614 (0.1)	0.474 (0.12)	0.5864 (0.11)
BL	0.5621 (0.1)	0.4529 (0.12)	0.4719 (0.14)	0.5871 (0.09)	0.4537 (0.12)	0.606 (0.1)

The first column represents the prior used in the model in both marker main effects and env:marker interaction term (when included in the model)

to this, some advantages of jointly modelling the multi-traits is that in this way the correlation among the traits is appropriately accounted for and can help to improve statistical power and the precision of parameter estimation, which are very important in genomic selection, because they can help to improve prediction accuracy and reduce trait selection bias (Schaeffer 1984; Pollak et al. 1984; Montesinos-López et al. 2018).

An example of this is when crop breeders collect phenotypic data for multiple traits such as grain yield and its components (grain type, grain weight, biomass, etc.), tolerance to biotic and abiotic stresses, and grain quality (taste, shape, color, nutrient, and/or content) (Montesinos-López et al. 2016). In this and many other cases, sometimes the interest is to predict traits that are difficult or expensive to measure with those that are easy to measure or the aim can be to improve all these correlated traits simultaneously (Henderson and Quaas 1976; Calus and Veerkamp 2011; Jiang et al. 2015). In these lines, there is evidence of the usefulness of multi-trait modelling. For example, Jia and Jannink (2012) showed that, compared to single-trait modelling, the prediction accuracy of low-heritability traits could be increased by using a multi-trait model when the degree of correlation between traits is at least moderate. They also found that multi-trait models had better prediction accuracy when phenotypes were not available on all individuals and traits. Joint modelling also has been found useful for increasing prediction accuracy when the traits of interest are not measured in the individuals of the testing set, but this and other traits were observed in individuals in the training set (Pszczola et al. 2013).

6.8.1 Genomic Multi-trait Linear Model

The genomic multi-trait linear model adopts a univariate genomic linear model structure for each trait but with correlated residuals and genotypic effects for traits in the same individual. Assuming that for individual j, n_T traits (Y_{jt}, $t = 1, \ldots, n_T$) are measured, this model assumes that

$$\begin{bmatrix} Y_{j1} \\ Y_{j2} \\ \vdots \\ Y_{jn_T} \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_{n_T} \end{bmatrix} + \begin{bmatrix} \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{\beta}_1 \\ \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{\beta}_{n_T} \end{bmatrix} + \begin{bmatrix} g_{j1} \\ g_{j2} \\ \vdots \\ g_{jn_T} \end{bmatrix} + \begin{bmatrix} \epsilon_{j1} \\ \epsilon_{j2} \\ \vdots \\ \epsilon_{jn_T} \end{bmatrix},$$

where μ_t, $t = 1, \ldots, n_T$, are the specific trait intercepts, $\boldsymbol{x}_j$ is a vector of covariates equal for all traits, g_{jt}, $t = 1, \ldots, n_T$, are the specific trait genotype effects, and ϵ_{jt}, $t = 1, \ldots, n_T$ are the random error terms corresponding to each trait. In matrix notation, it can be expressed as

$$\boldsymbol{Y}_j = \boldsymbol{\mu} + \boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}_j + \boldsymbol{g}_j + \boldsymbol{\epsilon}_j, \tag{6.8}$$

where $\boldsymbol{Y}_j = \left[Y_{j1}, \ldots, Y_{jn_T}\right]^{\mathrm{T}}$, $\boldsymbol{\mu} = \left[\mu_1, \ldots, \mu_{n_T}\right]^{\mathrm{T}}$, $\boldsymbol{B} = \left[\boldsymbol{\beta}_1, \quad \ldots, \quad \boldsymbol{\beta}_{n_T}\right]$, $\boldsymbol{g}_j = \left[g_{j1}, \ldots, g_{jn_T}\right]^{\mathrm{T}}$, and $\boldsymbol{\epsilon}_j = [\epsilon, \ldots, \epsilon_{n_T}]^{\mathrm{T}}$. The residual vectors are assumed to be independent with multivariate normal distribution, that is $\boldsymbol{\epsilon}_j \sim N_{n_T}(\boldsymbol{0}, \boldsymbol{R})$, and all the random genotype effects are assumed to be $\boldsymbol{g} = \left[\boldsymbol{g}_1^{\mathrm{T}}, \ldots, \boldsymbol{g}_J^{\mathrm{T}}\right]^{\mathrm{T}} \sim N(\boldsymbol{0}, \boldsymbol{G}\otimes\boldsymbol{\Sigma}_{\mathrm{T}})$, $\otimes$ being the Kronecker product. For a full Bayesian specification of this model, we suppose that $\boldsymbol{\beta} = \mathrm{vec}(\boldsymbol{B}) \sim N(\boldsymbol{\beta}_0, \boldsymbol{\Sigma}_\beta)$, that is, marginally, for the fixed effect of each trait, a prior multivariate normal distribution is adopted, $\boldsymbol{\beta}_t \sim N_p\left(\boldsymbol{\beta}_{t0}, \boldsymbol{\Sigma}_{\beta_t}\right), t = 1, \ldots, n_T$; a flat prior for the intercepts, $f(\boldsymbol{\mu}) \propto 1$; and independent inverse Wishart distributions for the covariance matrix of residuals $\boldsymbol{R}$ and for $\boldsymbol{\Sigma}_T$, that is, $\boldsymbol{\Sigma}_T \sim \mathrm{IW}(v_t, \boldsymbol{S}_t)$ and $\boldsymbol{R} \sim \mathrm{IW}(v_R, \boldsymbol{S}_R)$.

Putting all the information together where the measured traits of each individual ($\boldsymbol{Y}_j$) are accommodated in the rows of a matrix response ($\boldsymbol{Y}$), model (6.8) can be expressed as

$$\boldsymbol{Y} = \boldsymbol{1}_J\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{XB} + \boldsymbol{Z}_1\boldsymbol{b}_1 + \boldsymbol{E}, \tag{6.9}$$

where $=[\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_J]^{\mathrm{T}}$, $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_J]^{\mathrm{T}}$, $\boldsymbol{b}_1 = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_J]^{\mathrm{T}}$, and $\boldsymbol{E} = [\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_J]^{\mathrm{T}}$. Note that under this notation, $\boldsymbol{E}^{\mathrm{T}} \sim MN_{n_T\times J}(\boldsymbol{0}, \boldsymbol{R}, \boldsymbol{I}_J)$ or equivalently $\boldsymbol{E} \sim MN_{J\times n_T}(\boldsymbol{0}, \boldsymbol{I}_J, \boldsymbol{R})$, and $\boldsymbol{b}_1^{\mathrm{T}} \sim MN_{n_T\times J}(\boldsymbol{0}, \boldsymbol{\Sigma}_T, \boldsymbol{G})$ or $\boldsymbol{b}_1 \sim MN_{J\times n_T}(\boldsymbol{0}, \boldsymbol{G}, \boldsymbol{\Sigma}_T)$. Here $\boldsymbol{Z} \sim MN_{J\times n_T}(\boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V})$ means that the random matrix $\boldsymbol{Z}$ follows the matrix variate normal distribution with parameters $\boldsymbol{M}$, $\boldsymbol{U}$, and $\boldsymbol{V}$, or equivalently, that the Jn_T random vector $\mathrm{vec}(\boldsymbol{Z})$ is distributed as $N_{Jn_T}(\mathrm{vec}(\boldsymbol{M}), \boldsymbol{V}\otimes\boldsymbol{U})$, with $\mathrm{vec}(\cdot)$ denoting the vectorization of a matrix that stacks the columns of this in a single column. Note that when $\boldsymbol{\Sigma}_T$ and $\boldsymbol{R}$ are diagonal matrices, model (6.9) is equivalent to separately fitting a univariate GBLUP model to each trait.

The conditional distribution of all traits is given by

$$\begin{aligned} f(\boldsymbol{Y}|\boldsymbol{\mu},\boldsymbol{\beta},\boldsymbol{b}_1,\boldsymbol{\Sigma}_T,\mathbf{R}) &= \frac{|\boldsymbol{R}|^{-\frac{J}{2}}}{(2\pi)^{Jn_T}}\exp\left\{-\frac{1}{2}\mathrm{tr}\left[\boldsymbol{R}^{-1}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\mathrm{T}}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)^{\mathrm{T}}\boldsymbol{I}_J\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\mathrm{T}}-\boldsymbol{X\beta}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)\right]\right\} \\ &= \frac{|\boldsymbol{R}|^{-\frac{J}{2}}}{(2\pi)^{Jn_T}}\exp\left\{-\frac{1}{2}\left[\mathrm{vec}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\mathrm{T}}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)\right]^{\mathrm{T}}\left(\boldsymbol{R}^{-1}\otimes\boldsymbol{I}_J\right)\left[\mathrm{vec}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\mathrm{T}}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)\right]\right\} \end{aligned}$$

and the joint posterior of parameters $\boldsymbol{\mu}$, $\boldsymbol{B}$, $\boldsymbol{b}_1$, $\boldsymbol{\Sigma}_T$, and $\boldsymbol{R}$ is given by

$$f(\boldsymbol{\mu},\boldsymbol{B},\boldsymbol{b}_1,\boldsymbol{\Sigma}_T,\boldsymbol{R}|\boldsymbol{Y}) \propto f(\boldsymbol{Y}|\boldsymbol{\mu},\boldsymbol{B},\boldsymbol{b}_1,\boldsymbol{\Sigma}_T,\mathbf{R})f(\boldsymbol{b}_1|\boldsymbol{\Sigma}_T)f(\boldsymbol{\Sigma}_T)f(\boldsymbol{\beta})f(\boldsymbol{R}),$$

where $f(\boldsymbol{b}_1|\boldsymbol{\Sigma}_T)$ denotes the conditional distribution of the genotype effects, and $f(\boldsymbol{\Sigma}_T)$, $f(\boldsymbol{\beta})$, and $f(\boldsymbol{R})$ denote the prior density distribution of $\boldsymbol{\Sigma}_T$, $\boldsymbol{B}$, and $\boldsymbol{R}$, respectively. This joint posterior distribution of the parameters doesn't have closed form; for this reason, next are derived the full conditional distributions for Gibbs sampler implementation.

Let $\boldsymbol{\beta}_0$ and $\boldsymbol{\Sigma}_\beta$ be the prior mean and variance of $\boldsymbol{\beta} = \text{vec}(\boldsymbol{B})$. Because $\text{tr}(\boldsymbol{AB}) = \text{vec}(\boldsymbol{A}^{\text{T}})^{\text{T}}\text{vec}(\boldsymbol{B}) = \text{vec}(\boldsymbol{B})^{\text{T}}\text{vec}(\boldsymbol{A}^{\text{T}})$ and $\text{vec}(\boldsymbol{AXB}) = (\boldsymbol{B}^{\text{T}} \otimes \boldsymbol{A})\text{vec}(\boldsymbol{X})$, we have that

$$
\begin{aligned}
P(\boldsymbol{\beta}|-) &\propto f(\boldsymbol{Y}|\boldsymbol{\mu},\boldsymbol{B},\boldsymbol{b}_1,\boldsymbol{\Sigma}_T,\mathbf{R})f(\boldsymbol{\beta}) \\
&\propto \exp\left\{\begin{array}{l} -\frac{1}{2}\left[\text{vec}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)-(\boldsymbol{I}_{n_T}\otimes\boldsymbol{X})\text{vec}(\boldsymbol{B})\right]^{\text{T}}\left(\boldsymbol{R}^{-1}\otimes\boldsymbol{I}_J\right) \\ \times\left[\text{vec}(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{Z}_1\boldsymbol{b}_1)-(\boldsymbol{I}_{n_T}\otimes\boldsymbol{X})\text{vec}(\boldsymbol{B})\right]-\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)^{\text{T}}\boldsymbol{\Sigma}_\beta^{-1}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)\end{array}\right\} \\
&\propto \exp\left\{\begin{array}{l} -\frac{1}{2}\left[\text{vec}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)-(\boldsymbol{I}_{n_T}\otimes\boldsymbol{X})\boldsymbol{\beta}\right]^{\text{T}}\left(\boldsymbol{R}^{-1}\otimes\boldsymbol{I}_J\right) \\ \times\left[\text{vec}(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{Z}_1\boldsymbol{b}_1)-(\boldsymbol{I}_{n_T}\otimes\boldsymbol{X})\boldsymbol{\beta}\right]-\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)^{\text{T}}\boldsymbol{\Sigma}_\beta^{-1}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)\end{array}\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}_0\right]^{\text{T}}\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}^{-1}\left[\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}_0\right]\right\},
\end{aligned}
$$

where $\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}} = \left[\boldsymbol{\Sigma}_\beta^{-1} + \left(\boldsymbol{R}^{-1}\otimes\boldsymbol{X}^{\text{T}}\boldsymbol{X}\right)\right]^{-1}$ and $\widetilde{\boldsymbol{\beta}}_0 = \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}\left[\boldsymbol{\Sigma}_\beta^{-1}\boldsymbol{\beta}_0 + \left(\boldsymbol{R}^{-1}\otimes\boldsymbol{X}^{\text{T}}\right)\text{vec}(\boldsymbol{Y}-\mathbf{1}_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{Z}_1\boldsymbol{b}_1)\right]$. So, the full conditional distribution of $\boldsymbol{\beta}$ is $N_p\left(\widetilde{\boldsymbol{\beta}}_0,\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}\right)$. Similarly, the full conditional distribution of $\boldsymbol{g} = \text{vec}(\boldsymbol{b}_1)$ is $N_J\left(\widetilde{\boldsymbol{g}},\widetilde{\boldsymbol{G}}\right)$, with $\widetilde{\boldsymbol{G}} = \left[\left(\boldsymbol{\Sigma}_T^{-1}\otimes\boldsymbol{G}^{-1}\right)+\left(\boldsymbol{R}^{-1}\otimes\boldsymbol{Z}_1^{\text{T}}\boldsymbol{Z}_1\right)\right]^{-1}$ and $\widetilde{\boldsymbol{g}} = \widetilde{\boldsymbol{G}}\left(\boldsymbol{R}^{-1}\otimes\boldsymbol{Z}_1^{\text{T}}\right)\text{vec}(\boldsymbol{Y}-\mathbf{1}_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{XB})$. Now, because $\text{vec}(\mathbf{1}_J\boldsymbol{\mu}^{\text{T}}) = (\boldsymbol{I}_{n_T}\otimes\mathbf{1}_J)\boldsymbol{\mu}$, similarly as before, the full conditional of $\boldsymbol{\mu}$ is $N_{n_T}\left(\widetilde{\boldsymbol{\mu}},\widetilde{\boldsymbol{\Sigma}}_\mu\right)$, where $\widetilde{\boldsymbol{\Sigma}}_\mu = J^{-1}\boldsymbol{R}$ and $\widetilde{\boldsymbol{\mu}} = \widetilde{\boldsymbol{\Sigma}}_\mu\left(\boldsymbol{R}^{-1}\otimes\mathbf{1}_J\right)\text{vec}(\boldsymbol{Y}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{B})$.

The full conditional distribution of $\boldsymbol{\Sigma}_T$

$$
\begin{aligned}
P(\boldsymbol{\Sigma}_T|-) &\propto P(\boldsymbol{b}_1|\boldsymbol{\Sigma}_T)P(\boldsymbol{\Sigma}_T) \\
&\propto |\boldsymbol{\Sigma}_T|^{-\frac{J}{2}}|\boldsymbol{G}|^{-\frac{n_T}{2}}\exp\left\{-\frac{1}{2}\text{tr}\left[\boldsymbol{b}_1^{\text{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1\boldsymbol{\Sigma}_T^{-1}\right]\right\}P(\boldsymbol{\Sigma}_T) \\
&\propto \boldsymbol{\Sigma}_T^{-\frac{\nu_T+J+n_T+1}{2}}\exp\left\{-\frac{1}{2}\text{tr}\left(\boldsymbol{b}_1^{\text{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1+\boldsymbol{S}_T\right)\boldsymbol{\Sigma}_T^{-1}\right\}.
\end{aligned}
$$

From here we have that $\boldsymbol{\Sigma}_T \mid - \sim \text{IW}\left(\nu_T + J, \boldsymbol{b}_1^{\text{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1 + \boldsymbol{S}_T\right)$. Now, because

$$
\begin{aligned}
P(\boldsymbol{R}|-) &\propto f(\boldsymbol{Y}|\boldsymbol{\mu},\boldsymbol{B},\boldsymbol{b}_1,\boldsymbol{\Sigma}_T,\mathbf{R})f(\boldsymbol{R}) \\
&\propto |\boldsymbol{R}|^{-\frac{J}{2}}\exp\left\{-\frac{1}{2}\text{tr}\left[\boldsymbol{R}^{-1}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)^{\text{T}}\boldsymbol{I}_J\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)\right]\right\} \\
&\quad |\boldsymbol{R}|^{-\frac{\nu_R+n_T+1}{2}}\exp\left\{-\frac{1}{2}\text{tr}\left(\boldsymbol{S}_T\boldsymbol{R}^{-1}\right)\right\} \\
&\propto |\boldsymbol{R}|^{-\frac{\nu_R+J+n_T+1}{2}}\exp\left\{-\frac{1}{2}\text{tr}\left[\boldsymbol{S}_T+\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)^{\text{T}}\left(\boldsymbol{Y}-1_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)\right]\boldsymbol{R}^{-1}\right\}
\end{aligned}
$$

the full conditional distribution of $\boldsymbol{R}$ is $\text{IW}\left(\widetilde{\nu}_R,\widetilde{\boldsymbol{S}}_R\right)$, where $\widetilde{\nu}_R = \nu_R + J$ and $\widetilde{\boldsymbol{S}}_R = \boldsymbol{S}_T + \left(\boldsymbol{Y}-\mathbf{1}_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{XB}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)^{\text{T}}\left(\boldsymbol{Y}-\mathbf{1}_J\boldsymbol{\mu}^{\text{T}}-\boldsymbol{X}\boldsymbol{\beta}-\boldsymbol{Z}_1\boldsymbol{b}_1\right)$.

In summary, a Gibbs sampler exploration of the joint posterior distribution of $\boldsymbol{\mu}$, $\boldsymbol{\beta}$, $\boldsymbol{g}$, $\boldsymbol{\Sigma}_T$, and $\boldsymbol{R}$ can be done with the following steps:

1. Simulate $\boldsymbol{\beta}$ from a multivariate normal distribution $N_p\left(\widetilde{\boldsymbol{\beta}}_0, \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}\right)$, where $\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}} = \left[\boldsymbol{\Sigma}_{\beta}^{-1} + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\right]^{-1}$ and $\widetilde{\boldsymbol{\beta}}_0 = \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}\left[\boldsymbol{\Sigma}_{\beta}^{-1}\boldsymbol{\beta}_0 + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{X}^{\mathrm{T}}\right)\mathrm{vec}(\boldsymbol{Y} - \mathbf{1}_J\boldsymbol{\mu}^{\mathrm{T}} - \boldsymbol{Z}_1\boldsymbol{b}_1)\right]$.
2. Simulate $\boldsymbol{\mu}$ from $N_{n_T}\left(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}}_{\mu}\right)$, where $\widetilde{\boldsymbol{\Sigma}}_{\mu} = J^{-1}\boldsymbol{R}$ and $\widetilde{\boldsymbol{\mu}} = \widetilde{\boldsymbol{\Sigma}}_{\mu}\left(\boldsymbol{R}^{-1} \otimes \mathbf{1}_J\right)$ $\mathrm{vec}(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{B} - \boldsymbol{Z}_1\boldsymbol{B})$.
3. Simulate $\boldsymbol{g} = \mathrm{vec}(\boldsymbol{b}_1)$ from $N_J\left(\widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{G}}\right)$, where $\widetilde{\boldsymbol{G}} = \left[\left(\boldsymbol{\Sigma}_T^{-1} \otimes \boldsymbol{G}^{-1}\right) + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{Z}_1^{\mathrm{T}}\boldsymbol{Z}_1\right)\right]^{-1}$ and $\widetilde{\boldsymbol{g}} = \widetilde{\boldsymbol{G}}\left(\boldsymbol{R}^{-1} \otimes \boldsymbol{Z}_1^{\mathrm{T}}\right)\mathrm{vec}(\boldsymbol{Y} - \mathbf{1}_J\boldsymbol{\mu}^{\mathrm{T}} - \boldsymbol{X}\boldsymbol{B})$.
4. Simulate $\boldsymbol{\Sigma}_T$ from $\mathrm{IW}\left(v_T + J, \boldsymbol{b}_1^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1 + \boldsymbol{S}_T\right)$.
5. Simulate $\boldsymbol{R}$ from $\mathrm{IW}\left(\widetilde{v}_R, \widetilde{\boldsymbol{S}}_R\right)$, where $\widetilde{v}_R = v_R + J$ and $\widetilde{\boldsymbol{S}}_R = \boldsymbol{S}_T + (\boldsymbol{Y} - \mathbf{1}_J\boldsymbol{\mu}^{\mathrm{T}} - \boldsymbol{X}\boldsymbol{B} - \boldsymbol{Z}_1\boldsymbol{b}_1)^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{1}_J\boldsymbol{\mu}^{\mathrm{T}} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}_1\boldsymbol{b}_1)$.
6. Return to step 1 or terminate when chain length is adequate to meet convergence diagnostics and the required sample size is reached.

An implementation of this model can be done using the github version of the BGLR R library, which can be accessed from https://github.com/gdlc/BGLR-R and can be installed directly in the R console by running the following commands: *install.packages('devtools'); library(devtools); install_git('https://github.com/gdlc/BGLR-R')*. This implementation also uses a flat prior for the fixed effect regression coefficients $\boldsymbol{\beta}$, and in such a case, the corresponding full conditional of this parameter is the same as step 1 of the Gibbs sampler given before, but removing $\boldsymbol{\Sigma}_{\beta}^{-1}$ and $\boldsymbol{\Sigma}_{\beta}^{-1}\boldsymbol{\beta}_0$ from $\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}$ and $\widetilde{\boldsymbol{\beta}}_0$, respectively. Specifically, model (6.8) can be implemented with this version of the BGLR package as follows:

```
ETA = list( list( X=X, model='FIXED' ), list( K=Z1GZ1', model='RKHS' ) )
A   = Multitrait(y = Y, ETA=ETA, resCov = list( type = 'UN', S0 = S_R, df0 =
v_R ), nIter = nI, burnIn = nb)
```

The first argument in the Multitrait function is the response variable which many times is a phenotype matrix where each row corresponds to the measurement of n_T traits in each individual. The second argument is a list predictor in which the first sub-list specifies the design matrix and prior model to the fixed effects part of the predictor in model (6.9), and in the second sub-list are specified the parameters of the distribution of random genetic effects of $\boldsymbol{b}_1$, where it is specified the $K = \boldsymbol{G}$ genomic relationship matrix, that accounts for the similarity between individuals based on marker information, df0 $= \boldsymbol{v}_T$ and $S0 = \boldsymbol{S}_T$ are the degrees of freedom parameter (v_T) and the scale matrix parameter ($\boldsymbol{S}_T$) of the inverse Wishart prior distribution for $\boldsymbol{\Sigma}_T$, respectively. In the third argument (resCOV), $S0$ and df0 are the scale matrix parameter ($\boldsymbol{S}_R$) and the degree of freedom parameter (v_R) of the inverse Wishart

prior distribution for $\boldsymbol{R}$. The last two arguments are the required number of iterations (**nI**) and the burn-in period (**nb**) for running the Gibbs sampler.

Similarly to the univariate case, model (6.9) can be equivalently described and implemented as a multivariate Ridge regression model, as follows:

$$\boldsymbol{Y} = \mathbf{1}_J\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{XB} + \boldsymbol{X}_1\boldsymbol{B}_1 + \boldsymbol{E}, \tag{6.10}$$

where $\boldsymbol{X}_1 = \boldsymbol{Z}_1\boldsymbol{L}_G$, $\boldsymbol{G} = \boldsymbol{L}_G\boldsymbol{L}_G^{\mathrm{T}}$ is the Cholesky factorization of $\boldsymbol{G}$, $\boldsymbol{B}_1 = \boldsymbol{L}_G^{-1}\boldsymbol{b}_1 \sim MN_{J\times n_T}(\mathbf{0}, \boldsymbol{I}_J, \boldsymbol{\Sigma}_T)$, and the specifications for the rest of parameters and prior distribution are the same as given in model (6.8). A Gibbs sampler implementation of this model is very similar to the one described before, with little modification. Indeed, a Gibbs implementation with the same multi-trait function is as follows:

```
L_G= t(chol(G))
X_1= Z_1L_G
ETA = list( list( X=X, model='FIXED' ), list( X=X_1, model='BRR' ) )
A   = Multitrait(y = Y, ETA=ETA, resCov = list( type = 'UN', S0 = S_R, df0 =
v_R ), nIter = nI, burnIn = nb)
```

with the only change in the second sub-list predictor, where now the design matrix $\boldsymbol{X}_1$ and the Ridge regression model (**BRR**) are specified.

Example 3 To illustrate the performance in terms of the prediction power of these models and how to implement this in R software, we considered a reduced data set that consisted of 50 wheat lines grown in two environments. In each individual, two traits were measured: FLRSDS and MIXTIM. The evaluation was done with a five-fold cross-validation, where lines were evaluated in some environments with all traits but are missing for all traits in other environments. Model (6.9) was fitted and the environment effect was assumed a fixed effect.

The results are shown in Table 6.4, where the average (standard deviation) of two performance criteria is shown for each trait in each environment: average Pearson's correlation (PC) and the mean squared error of prediction (MSE). Table 6.4 shows good correlation performance in both traits and in both environments, and better predictions were obtained in environment 2 with both criteria. The magnitude of the

Table 6.4 Average Pearson's correlation (PC) and mean squared error of prediction (MSE) between predicted and observed values across five random partitions where lines were evaluated in some environments with all traits but are missing for all traits in other environments, SD represent standard deviation across partitions

Env	Trait	PC (SD)	MSE (SD)
1	FLRSDS	0.6252 (0.276)	4.3009 (2.343)
	MIXTIM	0.6709 (0.367)	0.457 (0.343)
2	FLRSDS	0.6895 (0.219)	3.9553 (2.171)
	MIXTIM	0.7523 (0.157)	0.3764 (0.249)

MSE in the first trait is mainly because the measurement scale is greater than in the second trait.

The R codes to reproduce these results (Table 6.4) are shown in Appendix 5.

6.9 Bayesian Genomic Multi-trait and Multi-environment Model (BMTME)

Model (6.9) does not take into account the possible trait–genotype–environment interaction ($T \times G \times E$), when environment information is available. An extension of this model is the one proposed by Montesinos-López et al. (2016), who added this interaction term to vary the specific trait genetic effects ($\boldsymbol{g}_j$) across environments. If the information of n_{T} traits of J lines is collected in I environments, this model is given by

$$\boldsymbol{Y} = \boldsymbol{1}_{IJ}\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{XB} + \boldsymbol{Z}_1\boldsymbol{b}_1 + \boldsymbol{Z}_2\boldsymbol{b}_2 + \boldsymbol{E}, \tag{6.11}$$

where $=[\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_{IJ}]^{\mathrm{T}}$, $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{IJ}]^{\mathrm{T}}$, $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ are the incident lines and the incident environment–line interaction matrices, $\boldsymbol{b}_1 = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_J]^{\mathrm{T}}$, $\boldsymbol{b}_2 = [\boldsymbol{g}_{21}, \ldots, \boldsymbol{g}_{2IJ}]^{\mathrm{T}}$, and $\boldsymbol{E} = [\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_{IJ}]^{\mathrm{T}}$. Here, $\boldsymbol{b}_2 \mid \boldsymbol{\Sigma}_T, \boldsymbol{\Sigma}_E \sim MN_{IJ\times n_T}(\boldsymbol{0}, \boldsymbol{\Sigma}_E \otimes \boldsymbol{G}, \boldsymbol{\Sigma}_T)$, and similar to model (6.2), $\boldsymbol{b}_1 \mid \boldsymbol{\Sigma}_T \sim MN_{J\times n_T}(\boldsymbol{0}, \boldsymbol{G}, \boldsymbol{\Sigma}_T)$ and $\boldsymbol{E} \sim MN_{IJ\times n_T}(\boldsymbol{0}, \boldsymbol{I}_{IJ}, \boldsymbol{R})$. The complete Bayesian specification of this model assumes independent multivariate normal distributions for the columns of $\boldsymbol{B}$, that is, for the fixed effect of each trait a prior multivariate normal distribution is adopted, $\boldsymbol{\beta}_t \sim N_p(\boldsymbol{\beta}_{t0}, \boldsymbol{\Sigma}_{\beta_t}), t = 1, \ldots, n_T$; a flat prior for the intercepts, $f(\boldsymbol{\mu}) \propto 1$; and independent inverse Wishart distributions for the covariance matrices of residuals $\boldsymbol{R}$ and for $\boldsymbol{\Sigma}_T$, $\boldsymbol{\Sigma}_T \sim \mathrm{IW}(v_T, \boldsymbol{S}_T)$ and $\boldsymbol{R} \sim \mathrm{IW}(v_R, \boldsymbol{S}_R)$, and also an inverse Wishart distribution for $\boldsymbol{\Sigma}_E$, $\boldsymbol{\Sigma}_E \sim \mathrm{IW}(v_E, \boldsymbol{S}_E)$.

The full conditional distributions of $\boldsymbol{\mu}, \boldsymbol{B}, \boldsymbol{b}_1, \boldsymbol{b}_2$, and $\boldsymbol{R}$ can be derived as in model (6.9). The full conditional distribution of $\boldsymbol{\Sigma}_T$ is

$$\begin{aligned}
f(\boldsymbol{\Sigma}_T|-) &\propto f(\boldsymbol{b}_1|\boldsymbol{\Sigma}_T)P(\boldsymbol{b}_2|\boldsymbol{\Sigma}_T, \boldsymbol{\Sigma}_E)P(\boldsymbol{\Sigma}_T) \\
&\propto |\boldsymbol{\Sigma}_T|^{-\frac{J}{2}}|\boldsymbol{G}|^{-\frac{I}{2}} \exp\left\{-\frac{1}{2}\mathrm{tr}\left[\boldsymbol{b}_1^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1\boldsymbol{\Sigma}_T^{-1}\right]\right\} \\
&\quad \times |\boldsymbol{\Sigma}_T|^{-\frac{IJ}{2}}|\boldsymbol{\Sigma}_E \otimes \boldsymbol{G}|^{-\frac{n_T}{2}} \exp\left\{-\frac{1}{2}\mathrm{tr}\left[\boldsymbol{b}_2^{\mathrm{T}}\left(\boldsymbol{\Sigma}_E^{-1} \otimes \boldsymbol{G}^{-1}\right)\boldsymbol{b}_2\boldsymbol{\Sigma}_T^{-1}\right]\right\} P(\boldsymbol{\Sigma}_T) \\
&\propto \boldsymbol{\Sigma}_T^{-\frac{v_T+J+IJ+n_T+1}{2}} \exp\left\{-\frac{1}{2}\mathrm{tr}\left(\boldsymbol{b}_1^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1 + \boldsymbol{b}_2^{\mathrm{T}}\left(\boldsymbol{\Sigma}_E^{-1} \otimes \boldsymbol{G}^{-1}\right)\boldsymbol{b}_2 + \boldsymbol{S}_t\right)\boldsymbol{\Sigma}_T^{-1}\right\},
\end{aligned}$$

that is, $\boldsymbol{\Sigma}_T \mid - \sim \mathrm{IW}\left(v_T + J + IJ, \boldsymbol{b}_1^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1 + \boldsymbol{b}_2^{\mathrm{T}}\left(\boldsymbol{\Sigma}_E^{-1} \otimes \boldsymbol{G}^{-1}\right)\boldsymbol{b}_2 + \boldsymbol{S}_T\right)$.

Now, let be $\boldsymbol{b}_2^*$ a $Jn_T \times I$ matrix such that $\text{vec}\left(\boldsymbol{b}_2^{\text{T}}\right) = \text{vec}\left(\boldsymbol{b}_2^*\right)$. Then because $\boldsymbol{b}_2^{\text{T}} \mid \boldsymbol{\Sigma}_T, \boldsymbol{\Sigma}_E \sim MN_{n_T \times IJ}(\boldsymbol{0}, \boldsymbol{\Sigma}_T, \boldsymbol{\Sigma}_E \otimes \boldsymbol{G})$, $\text{vec}\left(\boldsymbol{b}_2^{\text{T}}\right) \mid \boldsymbol{\Sigma}_T, \boldsymbol{\Sigma}_E \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}_E \otimes (\boldsymbol{G} \otimes \boldsymbol{\Sigma}_{\text{T}}))$, and $\boldsymbol{b}_2^* \mid \boldsymbol{\Sigma}_T, \boldsymbol{\Sigma}_E \sim MN_{n_T \times IJ}(\boldsymbol{0}, \boldsymbol{G} \otimes \boldsymbol{\Sigma}_{\text{T}}, \boldsymbol{\Sigma}_E)$, the full conditional posterior distribution of $\boldsymbol{\Sigma}_E$ is

$$
\begin{aligned}
P(\boldsymbol{\Sigma}_E|\text{ELSE}) &\propto P(\boldsymbol{b}_2|\boldsymbol{\Sigma}_E)\, P(\boldsymbol{\Sigma}_E) \\
&\propto |\boldsymbol{\Sigma}_E|^{-\frac{JI}{2}} |\boldsymbol{G} \otimes \boldsymbol{\Sigma}_T|^{-\frac{I}{2}} \exp\left\{-\frac{1}{2}\text{tr}\left[\boldsymbol{\Sigma}_E^{-1}\boldsymbol{b}_2^{*\text{T}}\left(\boldsymbol{G}^{-1} \otimes \boldsymbol{\Sigma}_T^{-1}\right)\boldsymbol{b}_2^*\right]\right\} \\
&\quad |\boldsymbol{S}_E|^{\frac{\upsilon_E+I-1}{2}} \times |\boldsymbol{\Sigma}_E|^{-\frac{\upsilon_E}{2}} \exp\left\{-\frac{1}{2}\text{tr}\left(\boldsymbol{S}_E\boldsymbol{\Sigma}_E^{-1}\right)\right\} \\
&\propto |\boldsymbol{\Sigma}_E|^{-\frac{\upsilon_E+JL+I+1}{2}} \exp\left\{-\frac{1}{2}\text{tr}\left[\left(\boldsymbol{b}_2^{*\text{T}}\left(\boldsymbol{G}^{-1} \otimes \boldsymbol{\Sigma}_T^{-1}\right)\boldsymbol{b}_2^* + \boldsymbol{S}_E\right)\right]\boldsymbol{\Sigma}_E^{-1}\right\}
\end{aligned}
$$

which means that $\boldsymbol{\Sigma}_E \mid - \sim \text{IW}\left(\upsilon_E + JL, \boldsymbol{b}_2^{*\text{T}}\left(\boldsymbol{G}^{-1} \otimes \boldsymbol{\Sigma}_T^{-1}\right)\boldsymbol{b}_2^* + \boldsymbol{S}_E\right)$.

A Gibbs sampler to explore the joint posterior distribution of parameters of model (6.11), $\boldsymbol{\mu}$, $\boldsymbol{\beta}$, $\boldsymbol{b}_1$,$\boldsymbol{b}_2$, $\boldsymbol{\Sigma}_T$, $\boldsymbol{\Sigma}_E$, and $\boldsymbol{R}$, can be implemented with the following steps:

1. Simulate $\boldsymbol{\beta}$ from a multivariate normal distribution $N_p\left(\widetilde{\boldsymbol{\beta}}_0, \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}\right)$, where $\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}} = \left[\boldsymbol{\Sigma}_\beta^{-1} + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{X}^{\text{T}}\boldsymbol{X}\right)\right]^{-1}$ and $\widetilde{\boldsymbol{\beta}}_0 = \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{\beta}}\left[\boldsymbol{\Sigma}_\beta^{-1}\boldsymbol{\beta}_0 + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{X}^{\text{T}}\right)\right.$ $\left.\text{vec}(\boldsymbol{Y} - \boldsymbol{1}_{IJ}\boldsymbol{\mu}^{\text{T}} - \boldsymbol{Z}_1\boldsymbol{b}_1 - \boldsymbol{Z}_2\boldsymbol{b}_2)\right]$.
2. Simulate $\boldsymbol{\mu}$ from $N_{n_T}\left(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\Sigma}}_\mu\right)$, where $\widetilde{\boldsymbol{\Sigma}}_\mu = (IJ)^{-1}\boldsymbol{R}$ and $\widetilde{\boldsymbol{\mu}} = \widetilde{\boldsymbol{\Sigma}}_\mu\left(\boldsymbol{R}^{-1} \otimes \boldsymbol{1}_{IJ}\right)$ $\text{vec}(\boldsymbol{Y} - \boldsymbol{XB} - \boldsymbol{Z}_1\boldsymbol{b}_1 - \boldsymbol{Z}_2\boldsymbol{b}_2)$.
3. Simulate $\boldsymbol{g}_1 = \text{vec}(\boldsymbol{b}_1)$ from $N_J\left(\widetilde{\boldsymbol{g}}_1, \widetilde{\boldsymbol{G}}\right)$, where $\widetilde{\boldsymbol{G}} = \left[\left(\boldsymbol{\Sigma}_T^{-1} \otimes \boldsymbol{G}^{-1}\right) + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{Z}_1^{\text{T}}\boldsymbol{Z}_1\right)\right]^{-1}$ and $\widetilde{\boldsymbol{g}} = \widetilde{\boldsymbol{G}}\left(\boldsymbol{R}^{-1} \otimes \boldsymbol{Z}_1^{\text{T}}\right)\text{vec}(\boldsymbol{Y} - \boldsymbol{1}_{IJ}\boldsymbol{\mu}^{\text{T}} - \boldsymbol{XB} - \boldsymbol{Z}_2\boldsymbol{b}_2)$.
4. Simulate $\boldsymbol{g}_2 = \text{vec}(\boldsymbol{b}_2)$ from $N_J\left(\widetilde{\boldsymbol{g}}_2, \widetilde{\boldsymbol{G}}_2\right)$, where $\widetilde{\boldsymbol{G}}_2 = \left[\left(\boldsymbol{\Sigma}_T^{-1} \otimes \boldsymbol{\Sigma}_E^{-1} \otimes \boldsymbol{G}^{-1}\right) + \left(\boldsymbol{R}^{-1} \otimes \boldsymbol{Z}_2^{\text{T}}\boldsymbol{Z}_2\right)\right]^{-1}$ and $\widetilde{\boldsymbol{g}}_2 = \widetilde{\boldsymbol{G}}_2\left(\boldsymbol{R}^{-1} \otimes \boldsymbol{Z}_2^{\text{T}}\right)\text{vec}(\boldsymbol{Y} - \boldsymbol{1}_{IJ}\boldsymbol{\mu}^{\text{T}} - \boldsymbol{XB} - \boldsymbol{Z}_1\boldsymbol{b}_1)$.
5. Simulate $\boldsymbol{\Sigma}_T$ from $\text{IW}\left(\upsilon_T + J + IJ, \boldsymbol{b}_1^{\text{T}}\boldsymbol{G}^{-1}\boldsymbol{b}_1 + \boldsymbol{b}_2^{\text{T}}\left(\boldsymbol{\Sigma}_E^{-1} \otimes \boldsymbol{G}^{-1}\right)\boldsymbol{b}_2 + \boldsymbol{S}_T\right)$.
6. Simulate $\boldsymbol{\Sigma}_E$ from $\text{IW}\left(\upsilon_E + JL, \boldsymbol{b}_2^{*\text{T}}\left(\boldsymbol{G}^{-1} \otimes \boldsymbol{\Sigma}_T^{-1}\right)\boldsymbol{b}_2^* + \boldsymbol{S}_E\right)$.
7. Simulate $\boldsymbol{R}$ from $\text{IW}\left(\widetilde{\nu}_R, \widetilde{\boldsymbol{S}}_R\right)$, where $\widetilde{\nu}_R = \nu_R + IJ$ and $\widetilde{\boldsymbol{S}}_R = \boldsymbol{S}_T + (\boldsymbol{Y} - \boldsymbol{1}_{IJ}\boldsymbol{\mu}^{\text{T}} - \boldsymbol{XB} - \boldsymbol{Z}_1\boldsymbol{b}_1 - \boldsymbol{Z}_2\boldsymbol{b}_2)^{\text{T}}(\boldsymbol{Y} - \boldsymbol{1}_{IJ}\boldsymbol{\mu}^{\text{T}} - \boldsymbol{X\beta} - \boldsymbol{Z}_1\boldsymbol{b}_1 - \boldsymbol{Z}_2\boldsymbol{b}_2)$.
8. Return to step 1 or terminate when chain length is adequate to meet convergence diagnostics and the required sample size is reached.

A similar Gibbs sampler is implemented in the BMTME R package, with the main difference, that this package does not allow specifying a general fixed effect design matrix $\boldsymbol{X}$, only the corresponding to the design matrix for the environment effects, and also the intercept vector $\boldsymbol{\mu}$ is ignored because it is included in the fixed

environment effects. Specifically, to fit model (6.11) where the only fixed effects to be taken into account are the environment's effects, the R code to implement this with the BMTME package is as follows:

```
XE   = model.matrix(~Env,data=dat_F)
Z1   = model.matrix(~0+GID,data=dat_F)
Lg    = t(chol(G))
Z1_a = Z1%*%Lg
Z2   = model.matrix(~0+GID:Env,data=dat_F)
G2   = kronecker(diag(dim(XE)[2])),Lg)
Z2_a = Z2%*%G2
A =  BMTME(Y = Y, X = XE, Z1 = Z1_a, Z2 = Z2_a, nIter = nI, burnIn = nb, thin =
2, bs = 50)
```

where **Y** is the matrix of response variables where each row corresponds to the measurement of n_T traits in each individual, **XE** is a design matrix for the environment effects, **Z1** is the incidence matrix of the genetics effects, **Z2** is the design matrix of the genetic–environment interaction effects, **nI** and **nb** are the required number of interactions and the burn-in period, and *bs* is the number of blocks to use internally to sample from vec($\boldsymbol{b}_2$).

Example 4 To illustrate how to implement this model with the BMTME R package, we considered the data in Example 2, but now the explored model includes the trait–genotype–environment interaction.

The average results of the prediction performance in terms of PC and MSE for implementing the same five-fold cross-validation used in Example 3 are shown in Table 6.5. These results show an improvement in terms of prediction performance with this model in all trait environments combinations and in both criteria (PC and MSE) to measure the prediction performance, except in trait MIXTIM and Env 2, where the MSE is slightly greater than the one obtained with model (6.9), which does not take into account the triple interaction (trait–genotype–environment).

The R code used to obtained these results is given in Appendix 5.

Table 6.5 Average Pearson's correlation (PC) and mean squared error of prediction (MSE) between predicted and observed values across five random partitions where lines were evaluated in some environments with all traits but are missing for all traits in other environments

Env	Trait	PC (SD)	MSE (SD)
1	FLRSDS	0.668 (0.243)	3.4483 (1.9)
	MIXTIM	0.6913 (0.35)	0.4255 (0.325)
2	FLRSDS	0.7218 (0.219)	3.2304 (2.034)
	MIXTIM	0.7667 (0.188)	0.3806 (0.287)

Appendix 1

- The probability density function (pdf) of the scaled inverse Chi-square distribution with v degrees of freedom and scale parameter S, $\chi^{-2}(v, S)$, is given by

$$f\left(\sigma^2; v, S\right) = \frac{\left(\frac{S}{2}\right)^{\frac{v}{2}}}{\Gamma\left(\frac{v}{2}\right)\left(\sigma^2\right)^{1+\frac{v}{2}}} \exp\left(-\frac{S}{2\sigma^2}\right).$$

  and the mean, mode, and variance of this distribution are given by $\frac{S}{v-2}$, $\frac{S}{v+2}$, and $\frac{2S^2}{(v-2)^2(v-4)}$, respectively.
- The pdf of the gamma distribution with shape parameter s and rate parameter r:

$$f_{S_\beta}(x; s, r) = \frac{r^s x^{s-1}}{\Gamma(s)} \exp(-rx).$$

  The mean, mode, and variance of this distribution are s/r, $(s-1)/r$, and s/r^2, respectively.
- The pdf of a beta distribution with mean μ and precision parameter ϕ ("dispersion" parameter ϕ^{-1}) is given by

$$f(x; \mu, \phi) = \frac{1}{B[\mu\phi, (1-\mu)\phi]} x^{\mu\phi-1}(1-x)^{(1-\mu)\phi-1},$$

  where the relation with the standard parameterization of this distribution, Beta (α, β), is

$$\mu = \frac{\alpha}{\alpha+\beta}, \phi = \alpha+\beta.$$

- The pdf of a Laplace distribution is

$$f(x; b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right), b > 0, -\infty \leq x \leq \infty.$$

  The mean and variance of this distribution are 0 and $2b^2$.
- A random matrix $\mathbf{\Sigma}$ of dimension $p \times p$ is distributed as inverse Wishart distribution with parameter v and $\boldsymbol{S}$, $\mathbf{\Sigma} \sim \text{IW}(v, \boldsymbol{S})$, if it has a density function

$$f(\mathbf{\Sigma}) = \frac{1}{2^{\frac{vp}{2}}\Gamma_p(v/2)} |\boldsymbol{S}|^{\frac{v}{2}} |\mathbf{\Sigma}|^{-\frac{v+p+1}{2}} \exp\left[-\frac{1}{2}\text{tr}\left(\boldsymbol{S}\mathbf{\Sigma}^{-1}\right)\right],$$

  where $\Gamma_p(v/2)$ is the multivariate gamma function, $v > 0$, and $\mathbf{\Sigma}$ and $\boldsymbol{S}$ are positive defined matrices. The mean matrix of this distribution is $\frac{\boldsymbol{S}}{v-p-1}$.

- A $(p \times q)$ random matrix $\boldsymbol{Z}$ follows the matrix normal distribution with matrix parameters $\boldsymbol{M}$ $(p \times q)$, $\boldsymbol{U}$ $(p \times p)$, and $\boldsymbol{V}(q \times q)$, $\boldsymbol{Z} \sim MN_{p,\,q}(\boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V})$, if it has density

$$f(\boldsymbol{Z}|\boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V}) = \frac{\exp\left\{-\frac{1}{2}\mathrm{tr}\left[\boldsymbol{V}^{-1}(\boldsymbol{Z}-\boldsymbol{M})^{\mathrm{T}}\boldsymbol{U}^{-1}(\boldsymbol{Z}-\boldsymbol{M})\right]\right\}}{(2\pi)^{pq/2}|\boldsymbol{V}|^{p/2}|\boldsymbol{U}|^{q/2}}.$$

Appendix 2: Setting Hyperparameters for the Prior Distributions of the BRR Model

The following rules are those used in Pérez and de los Campos (2014), and provide proper but weakly informative prior distributions. In general, this consists of assigning a certain proportion of the total variance of the phenotypes, to the different components of the model.

Specifically, for model (6.3), first the total variance of y is partitioned into two components: (1) the error and (2) the linear predictor:

$$\mathrm{Var}(y_j) = \mathrm{Var}\left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{\beta}_0\right) + \sigma^2$$

Therefore, the average of the variance of the individuals, called total variance, is equal to

$$\frac{1}{n}\sum_{j=1}^{n}\mathrm{Var}(y_j) = \frac{1}{n}\sum_{j=1}^{n}\mathrm{Var}\left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{\beta}_0\right) + \sigma^2 = \frac{1}{n}\mathrm{tr}\left(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)\sigma_\beta^2 + \sigma^2 = V_M + V_\epsilon.$$

Then, by setting R_1^2 as the proportion of the total variance $(\mathbf{V}_y)$, that is explained by markers a priori, $V_M = R_1^2\mathbf{V}_y$, and replacing σ_β^2 in V_M by its prior mode, $\frac{S_\beta}{v_\beta+2}$, we have that

$$\frac{1}{n}\mathrm{tr}\left(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)\left(\frac{S_\beta}{v_\beta+2}\right) = R_1^2\mathbf{V}_y.$$

From here, once we have set a value for v_β, the scale parameter is given by

$$S_\beta = \frac{R_1^2\mathbf{V}_y}{\frac{1}{n}\mathrm{tr}\left(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)}\left(v_\beta+2\right).$$

A commonly used value of the shape parameter is $v_\beta = 5$ and the value for the proportion of explained variances is $R_1^2 = 0.5$.

Because the model only has two predictors and R_1^2 was set as the proportion of the total variance that is explained by markers a priori, the corresponding proportion that is explained by error a priori is $R_2^2 = 1 - R_1^2$. Then, similar to what was done before, once there is a value for the shape parameter of the prior distribution of σ^2, v, the value of the scale parameter is given by

$$S = \left(1 - R_1^2\right)\mathbf{V}_y(v + 2).$$

By default, $v = 5$ is often used.

Appendix 3: R Code Example 1

```
rm(list=ls())
library(BGLR)
load('dat_ls_E1.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
#Marker data
dat_M = dat_ls$dat_M
dim(dat_M)

dat_F = transform(dat_F, GID = as.character(GID))
head(dat_F,5)

#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
XM = dat_M[Pos,]
XM = scale(XM)
dim(XM)

n = dim(dat_F)[1]
y = dat_F$y

#10 random partitions
K = 10
set.seed(1)
PT = replicate(K,sample(n,0.20*n))

#BRR
ETA_BRR = list(list(model='BRR',X=XM))
Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
```

```
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,ETA=ETA_BRR,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 yp_ts = A$yHat[Pos_tst]
 Tab$MSEP[k] = mean((y[Pos_tst]-yp_ts)^2)
}

#GBLUP
dat_M = scale(dat_M)
G = tcrossprod(XM)/dim(XM)[2]
dim(G)
#Matrix design of GIDs
Z = model.matrix(~0+GID,data=dat_F,xlev = list(GID=unique
(dat_F$GID)))
K_L = Z%*%G%*%t(Z)
ETA_GB = list(list(model='RKHS',K = K_L))
#Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,ETA=ETA_GB,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 yp_ts = A$yHat[Pos_tst]
 Tab$MSEP_GB[k] = mean((y[Pos_tst]-yp_ts)^2)
}

#BA
ETA_BA = list(list(model='BayesA',X=XM))
#Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,ETA=ETA_BA,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 yp_ts = A$yHat[Pos_tst]
 Tab$MSEP_BA[k] = mean((y[Pos_tst]-yp_ts)^2)
}

#BB
ETA_BB = list(list(model='BayesB',X=XM))
#Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,ETA=ETA_BB,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
```

```
 yp_ts = A$yHat[Pos_tst]
 Tab$MSEP_BB[k] = mean((y[Pos_tst]-yp_ts)^2)
}

#BC
ETA_BC = list(list(model='BayesC',X=XM))
#Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,ETA=ETA_BC,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 yp_ts = A$yHat[Pos_tst]
 Tab$MSEP_BC[k] = mean((y[Pos_tst]-yp_ts)^2)
}

#BL
ETA_BL = list(list(model='BL',X=XM))
#Tab = data.frame(PT = 1:K,MSEP = NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,ETA=ETA_BL,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 yp_ts = A$yHat[Pos_tst]
 Tab$MSEP_BL[k] = mean((y[Pos_tst]-yp_ts)^2)
}
Tab

#Mean and SD across the five partitions
apply(Tab[,-1],2,function(x)c(mean(x),sd(x)))
```

Appendix 4: R Code Example 2

```
rm(list=ls())
library(BGLR)
library(BMTME)
load('dat_ls_E2.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
dim(dat_F)
#Marker data
dat_M = dat_ls$dat_M
dim(dat_M)
```

```
dat_F = transform(dat_F, GID = as.character(GID))
head(dat_F,5)

#Matrix design for markers
Pos = match(dat_F$GID,row.names(dat_M))
XM = dat_M[Pos,]
dim(XM)
XM = scale(XM)
#Environment design matrix
XE = model.matrix(~0+Env,data=dat_F)[,-1]
head(XE)
#Environment-marker design matrix
XEM = model.matrix(~0+XM:XE)
#GID design matrix and Environment-GID design matrix
#for RKHS models
Z_L = model.matrix(~0+GID,data=dat_F,xlev = list(GID=unique
(dat_F$GID)))
Z_LE = model.matrix(~0+GID:Env,data=dat_F,
          xlev = list(GID=unique(dat_F$GID),Env = unique(dat_F$Env)))
#Genomic relationship matrix derived from markers
dat_M = scale(dat_M)
G = tcrossprod(dat_M)/dim(dat_M)[2]
dim(G)
#Covariance matrix for Zg
K_L = Z_L%*%G%*%t(Z_L)
#Covariance matrix for random effects ZEg
K_LE = Z_LE%*%kronecker(diag(4),G)%*%t(Z_LE)
n = dim(dat_F)[1]
y = dat_F$y

#Number of random partitions
K = 5
PT = CV.KFold(dat_F,DataSetID = 'GID', K=5, set_seed = 1)
Models = c('BRR','RKHS','BayesA','BayesB','BayesC','BL')
Tab = data.frame()
for(m in 1:6)
{
 ETA1 = list(list(model=Models[m],X=XM))
 ETA2 = list(list(model='FIXED',X=XE),list(model=Models[m],X=XM))
 ETA3 = list(list(model='FIXED',X=XE),list(model=Models[m],X=XM),
        list(model=Models[m],X=XEM))
 if(Models[m]=='RKHS')
 {
  ETA1 = list(list(model='RKHS',K=K_L))
  ETA2 = list(list(model='FIXED',X=XE),list(model='RKHS',K=K_L))
  ETA3 = list(list(model='FIXED',X=XE),list(model='RKHS',K=K_L),
         list(model='RKHS',K=K_LE))
 }
 Tab1_m = data.frame(PT = 1:K,MSEP = NA)
 set.seed(1)
 Tab2_m = Tab1_m
 Tab3_m = Tab2_m
 for(k in 1:K)
```

```
 {
  Pos_tst = PT$CrossValidation_list[[k]]
  y_NA = y
  y_NA[Pos_tst] = NA
  A = BGLR(y=y_NA,ETA=ETA1,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
  yp_ts = A$yHat[Pos_tst]
  Tab1_m$MSEP[k] = mean((y[Pos_tst]-yp_ts)^2)
  Tab1_m$Cor[k] = cor(y[Pos_tst],yp_ts)

  A = BGLR(y=y_NA,ETA=ETA2,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
  yp_ts = A$yHat[Pos_tst]
  Tab2_m$MSEP[k] = mean((y[Pos_tst]-yp_ts)^2)
  Tab2_m$Cor[k] = cor(y[Pos_tst],yp_ts)

  A = BGLR(y=y_NA,ETA=ETA3,nIter = 1e4,burnIn = 1e3,verbose = FALSE)
  yp_ts = A$yHat[Pos_tst]
  Tab3_m$MSEP[k] = mean((y[Pos_tst]-yp_ts)^2)
  Tab3_m$Cor[k] = cor(y[Pos_tst],yp_ts)
 }
 Tab = rbind(Tab,data.frame(Model=Models[m],Tab1_m,Tab2_m,Tab3_m))
}
Tab
```

Appendix 5

R Code Example 3

```
rm(list=ls(all=TRUE))
library(BGLR)
library(BMTME)
library(dplyr)

load('dat_ls.RData',verbose=TRUE)
dat_F = dat_ls$dat_F
head(dat_F)
Y = as.matrix(dat_F[,-(1:2)])
dat_F$Env = as.character(dat_F$DS)
G = dat_ls$G
J = dim(G)[1]
XE = matrix(model.matrix(~0+Env,data=dat_F)[,-1],nc=1)
Z = model.matrix(~0+GID,data=dat_F)
K = Z%*%G%*%t(Z)
#Partitions for a 5-FCV
PT_ls = CV.KFold(dat_FF, DataSetID='GID',K=5,set_seed = 123)
PT_ls = PT_ls$CrossValidation_list
#Predictor BGLR
ETA = list(list(X=XE,model='FIXED'),list(K=K,model='RKHS'))
#Function to summarize the performance prediction: PC_MM_f
source('PC_MM.R')#See below
Tab = data.frame()
```

```
set.seed(1)
for(p in 1:5)
{
 Y_NA = Y
 Pos_NA = PT_ls[[p]]
 Y_NA[Pos_NA,] = NA
 A = Multitrait(y = Y_NA, ETA=ETA,
           resCov = list( type = 'UN', S0 = diag(2), df0 = 5 ),
           nIter = 5e3, burnIn = 1e3)
 PC = PC_MM_f(Y[Pos_NA,],A$yHat[Pos_NA,],Env=dat_F$Env[Pos_NA])
 Tab = rbind(Tab,data.frame(PT=p,PC))
 cat('PT=',p,'\n')
}
Tab_R = Tab%>%group_by(Env,Trait)%>%select(Cor,MSEP)%>%summarise
(Cor_mean = mean(Cor),
                                Cor_sd   = sd(Cor),
                                MSEP_mean = mean(MSEP),
                                MSEP_sd = sd(MSEP))
Tab_R = as.data.frame(Tab_R)
Tab_R
#Save in the same folder the following in a file with name "PC_MM.R"
#Performance criteria
PC_MM_f<-function(y,yp,Env=NULL)
{
 if(is.null(Env))
 {
  Cor  = diag(cor(as.matrix(y),as.matrix(yp)))
  MSEP = colMeans((y-yp)^2)
  PC = data.frame(Trait = colnames(y),Cor=Cor, MSEP=MSEP)
 }
 else
 {
  PC = data.frame()
  Envs = unique(Env)
  nE = length(Envs)
  for(e in 1:nE)
  {
   y_e = y[Env==Envs[e],]
   yp_e = yp[Env==Envs[e],]
   Cor  = diag(cor(as.matrix(y_e),as.matrix(yp_e)))
   MSEP = colMeans((y_e-yp_e)^2)
   PC = rbind(PC,data.frame(Trait = colnames(y),Env=Envs[e],Cor=Cor,
MSEP=MSEP))
  }
 }
 PC
}
```

R Code for Example 4

```
rm(list=ls(all=TRUE))
library(BMTME)
library(dplyr)
load('dat_ls.RData',verbose=TRUE)
dat_F = dat_ls$dat_F
head(dat_F)
Y = as.matrix(dat_F[,-(1:2)])
dat_F$Env = as.character(dat_F$DS)
G = dat_ls$G
Lg = t(chol(G))
XE = model.matrix(~Env,data=dat_F)
Z1 = model.matrix(~0+GID,data=dat_F)
Z1_a = Z1%*%Lg
Z2 = model.matrix(~0+GID:Env,data=dat_F)
L2 = kronecker(diag(dim(XE)[2]),Lg)
Z2_a = Z2%*%L2
#Partitions for a 5-FCV
PT_ls = CV.KFold(dat_FF, DataSetID='GID',K=5,set_seed = 123)
PT_ls = PT_ls$CrossValidation_list
source('PC_MM.R')#See file R "PC_MM.R" defined in example 6.1
Tab = data.frame()
set.seed(1)
for(p in 1:5)
{
 Y_NA = Y
 Pos_NA = PT_ls[[p]]
 Y_NA[Pos_NA,] = NA
 A = BMTME(Y = Y_NA, X = XE, Z1 = Z1_a, Z2 = Z2_a,
           nIter = 3e3, burnIn = 5e2, thin = 2, bs = 50)
 PC = PC_MM_f(Y[Pos_NA,],A$yHat[Pos_NA,],Env=dat_F$Env[Pos_NA])
 Tab = rbind(Tab,data.frame(PT=p,PC))
 cat('PT=',p,'\n')
}
Tab_R = Tab%>%group_by(Env,Trait)%>%select(Cor,MSEP)%>%summarise
(Cor_mean = mean(Cor),
                                Cor_sd  = sd(Cor),
                                MSEP_mean = mean(MSEP),
                                MSEP_sd = sd(MSEP))
Tab_R = as.data.frame(Tab_R)
Tab_R
```

References

Box GEP, Tiao GC (1992) Bayesian inference in statistical analysis. Wiley, New York
Calus MP, Veerkamp RF (2011) Accuracy of multi-trait genomic selection using different methods. Genet Sel Evol 43(1):26
Casella G, George EI (1992) Explaining the Gibbs sampler. Am Stat 46(3):167–174
Christensen R, Johnson W, Branscum A, Hanson TE (2011) Bayesian ideas and data analysis: an introduction for scientists and statisticians. Chapman & Hall/CRC, Stanford, CA
de los Campos G, Hickey JM, Pong-Wong R, Daetwyler HD, Calus MP (2013) Whole-genome regression and prediction methods applied to plant and animal breeding. Genetics 193 (2):327–345
Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB (2013) Bayesian data analysis. Chapman and Hall/CRC, Stanford, CA
Habier D, Tetens J, Seefried FR, Lichtner P, Thaller G (2010) The impact of genetic relationship information on genomic breeding values in German Holstein cattle. Genet Sel Evol 42(1):5
Habier D, Fernando RL, Kizilkaya K, Garrick DJ (2011) Extension of the Bayesian alphabet for genomic selection. BMC Bioinformatics 12(1):186
Henderson C (1975) Best linear unbiased estimation and prediction under a selection model. Biometrics 31(2):423–447. https://doi.org/10.2307/2529430
Henderson CR, Quaas RL (1976) Multiple trait evaluation using relative's records. J Anim Sci 43: 11–88
Jia Y, Jannink JL (2012) Multiple-trait genomic selection methods increase genetic value prediction accuracy. Genetics 192(4):1513–1522
Jiang J, Zhang Q, Ma L, Li J, Wang Z, Liu JF (2015) Joint prediction of multiple quantitative traits using a Bayesian multivariate antedependence model. Heredity 115(1):29–36
Lehermeier C, Wimmer V, Albrecht T, Auinger HJ, Gianola D, Schmid VJ, Schön CC (2013) Sensitivity to prior specification in Bayesian genome-based prediction models. Stat Appl Genet Mol Biol 12(3):375–391
Meuwissen TH, Hayes BJ, Goddard ME (2001) Prediction of total genetic value using genome-wide dense marker maps. Genetics 157(4):1819–1829
Montesinos-López OA, Montesinos-López A, Crossa J, Toledo FH, Pérez-Hernández O, Eskridge KM, Rutkoski J (2016) A genomic Bayesian multi-trait and multi-environment model. G3 6 (9):2725–2744
Montesinos-López OA, Montesinos-López A, Montesinos-López JC, Crossa J, Luna-Vázquez FJ, Salinas-Ruiz J (2018) A Bayesian multiple-trait and multiple-environment model using the matrix normal distribution. In: Physical methods for stimulation of plant and mushroom development. IntechOpen, Croatia, p 19
Park T, Casella G (2008) The Bayesian lasso. J Am Stat Assoc 103(482):681–686
Pérez P, de los Campos G (2013) BGLR: a statistical package for whole genome regression and prediction. R package version 1(0.2)
Pérez P, de los Campos G (2014) Genome-wide regression and prediction with the BGLR statistical package. Genetics 198(2):483–495
Pérez P, de los Campos G, Crossa J, Gianola D (2010) Genomic-enabled prediction based on molecular markers and pedigree using the Bayesian linear regression package in R. Plant Genome 3(2):106–116
Pollak EJ, Van der Werf J, Quaas RL (1984) Selection bias and multiple trait evaluation. J Dairy Sci 67(7):1590–1595
Pszczola M, Veerkamp RF, De Haas Y, Wall E, Strabel T, Calus MPL (2013) Effect of predictor traits on accuracy of genomic breeding values for feed intake based on a limited cow reference population. Animal 7(11):1759–1768
Schaeffer LR (1984) Sire and cow evaluation under multiple trait models. J Dairy Sci 67 (7):1567–1580
VanRaden PM (2007) Genomic measures of relationship and inbreeding. Interbull Bull 7:33–36

Chapter 7
Bayesian and Classical Prediction Models for Categorical and Count Data

7.1 Introduction

Categorical and count response variables are common in many plant breeding programs. For this reason, in this chapter, the fundamental and practical issues for implementing genomic prediction models with these types of response variables are provided. Also, current open source software is used for its implementation. The prediction models proposed here are Bayesian and classical models. For each type of these models, we provide the fundamental principles and concepts, whereas their practical implementation is illustrated with examples in the genomic selection context. Taken into account are the most common terms in the predictor, such as the main effects of environments and genotypes, and interaction terms like genotype × environment interaction and marker × environment interaction. First, the models under the Bayesian framework are presented and then those under a classical framework. Next, the fundamentals and applications of the Bayesian ordinal regression model are provided.

7.2 Bayesian Ordinal Regression Model

In many applications, the response (Y) is not continuous, but rather multi-categorical in nature: ordinal (the categories of the response can be ordered, 1, . . ., C) or nominal (the categories have no order). For example, in plant breeding, categorical scores for resistance and susceptibility are often collected [disease: 1 (no disease), 2 (low infection), 3 (moderate infection), 4 (high infection), and 5 (complete infection)]. Another example is in animal breeding, where calving ease is scored as 1, 2, or 3 to indicate normal birth, slight difficult birth, or extreme difficult birth, respectively. The categorical traits are assigned to one of a set of mutually exclusive and exhaustive response values.

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_7

It is well documented that linear regression models in this situation are difficult to justify (Gianola 1980, 1982; Gianola and Foulley 1983) and inadequate results can be obtained, unless there is a large number of categories and the data seem to show a distribution that is approximately normal (Montesinos-López et al. 2015b).

For the ordinal model, we appeal to the existence of a latent continuous random variable and the categories are conceived as contiguous intervals on the continuous scale, as presented in McCullagh (1980), where the points of division (thresholds) are denoted as $\gamma_0, \gamma_1, \ldots, \gamma_C$. In this way, the ordinal model assumes that conditioned to $\boldsymbol{x}_i$ (covariates of dimension p), Y_i is a random variable that takes values $1, \ldots, C$, with the following probabilities:

$$\begin{aligned} p_{ic} &= P(Y_i = c) = P(\gamma_{c-1} \leq L_i \leq \gamma_c) \\ &= F\left(\gamma_c + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right) - F\left(\gamma_{c-1} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right), \quad c = 1, \ldots, C, \end{aligned} \tag{7.1}$$

where $L_i = -\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \epsilon_i$ is a latent variable, ϵ_i is a random error term with cumulative distribution function F, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^{\mathrm{T}}$ denotes the parameter vector associated with the effects of the covariates, and $-\infty = \gamma_0 < \gamma_1 < \ldots < \gamma_C = \infty$ are threshold parameters and also need to be estimated, that is, the parameter vector to be estimated in this model is $\boldsymbol{\theta} = \left(\boldsymbol{\beta}^{\mathrm{T}}, \boldsymbol{\gamma}^{\mathrm{T}}, \sigma_\beta^2\right)^{\mathrm{T}}$, where $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{C-1})^{\mathrm{T}}$.

All the ordinal models presented in this chapter share the property that the categories can be considered as contiguous intervals on some continuous scale, but they differ in their assumptions regarding the distributions of the latent variable. When F is the standard logistic distribution function, the resulting model is known as the **logistic** ordinal model:

$$p_{ic} = F\left(\gamma_c + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right) - F\left(\gamma_{c-1} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right), \quad c = 1, \ldots, C,$$

where $F(z) = \frac{1}{1+\exp(-z)}$. When F is the standard normal distribution function, the resulting model is the ordinal **probit** model:

$$p_{ic} = F\left(\gamma_c + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right) - F\left(\gamma_{c-1} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right), \quad c = 1, \ldots, C,$$

where $F(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right) du$. When the response value only takes two values, model (7.1) is reduced to the binary regression model, which in the first case is better known as logistic regression, while in the second case, it is known as the probit regression model. For this reason, logistic regression or probit regression is a particular case of ordinal regression even under a Bayesian framework; for this reason, the Bayesian frameworks that will be described for ordinal regression in this chapter are reduced to logistic or probit regression when only one threshold parameter needs to be estimated (γ_1), or equivalently this is set to 0 and an intercept parameter (β_0) is added to the vector coefficients that also need to be estimated,

$\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^{\mathrm{T}}$. In this model, the following development will be concentrated in the probit ordinal regression model.

A Bayesian specification for the ordinal regression is very similar to the linear model described in Chap. 6, and assumes the following prior distribution for the parameter vector $\boldsymbol{\theta} = \left(\boldsymbol{\beta}^T, \boldsymbol{\gamma}^T, \sigma_\beta^2\right)^{\mathrm{T}}$:

$$f(\boldsymbol{\theta}) \propto \frac{1}{\left(\sigma_\beta^2\right)^{\frac{p}{2}}} \exp\left(\left[-\frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}\right]\right) \frac{\left(\frac{S_\beta}{2}\right)^{\frac{\nu_\beta}{2}}}{\Gamma\left(\frac{\nu_\beta}{2}\right)\left(\sigma_\beta^2\right)^{1+\frac{\nu_\beta}{2}}} \exp\left(-\frac{S_\beta}{2\sigma_\beta^2}\right)$$

That is, a flat prior is assumed for the threshold parameter $\boldsymbol{\gamma}$, a multivariate normal distribution for the vector of beta coefficients, $\boldsymbol{\beta} \mid \sigma_\beta^2 \sim N_p\left(\mathbf{0}, \sigma_\beta^2 \boldsymbol{I}_p\right)$, and a scaled inverse chi-squared for σ_β^2. From now on, this model will be referred to as BRR in this chapter in analogy to the Bayesian linear regression and the way that this is implemented in the BGLR R package. Similar to the genomic linear regression model, the posterior distribution of the parameter vector does not have a known form and a Gibbs sampler is used to explore this; for this reason, in the coming lines, the Gibbs sampling method proposed by Albert and Chib (1993) is described. To make it possible to derive the full conditional distributions, the parameter vector is augmented with a latent variable in the representation of model (7.1).

The joint probability density function of the vector of observations, $\boldsymbol{Y} = (Y_1, \ldots, Y_n)^{\mathrm{T}}$, and the latent variables $\boldsymbol{L} = (L_1, \ldots, L_n)^{\mathrm{T}}$, evaluated in the vector values $\boldsymbol{y} = (y_1, \ldots, y_n)^{\mathrm{T}}$ and $\boldsymbol{l} = (l_1, \ldots, l_n)^{\mathrm{T}}$, is given by

$$\begin{aligned} f(\boldsymbol{y}, \boldsymbol{l} | \boldsymbol{\beta}, \boldsymbol{\gamma}) &= \prod_{i=1}^{n} f_{L_i}(l_i) I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i) \\ &= \prod_{i=1}^{n} \exp\left[-\frac{1}{2}\left(l_i + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right)^2\right] I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i) \\ &= \exp\left[-\frac{1}{2}\sum_{i=1}^{n}\left(l_i + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}\right)^2\right] \prod_{i=1}^{n} I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i), \end{aligned}$$

where I_{A} is the indicator function of a set A, and $f_{L_i}(l_i)$ is the density function of the latent variable that in the ordinal probit regression model corresponds to the normal distribution with mean $-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}$ and unit variance. Then the full conditional of the jth component of $\boldsymbol{\beta}$, β_j, given the rest of the parameters and the latent variables, is given by

$$
\begin{aligned}
f(\beta_j|-) &\propto f(\boldsymbol{y}, \boldsymbol{l}|\boldsymbol{\beta}, \boldsymbol{\gamma}) f\left(\boldsymbol{\beta}|\sigma_\beta^2\right) \\
&\propto \prod_{i=1}^{n} f_{L_i}(l_i) I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i) f\left(\boldsymbol{\beta}|\sigma_\beta^2\right) \\
&\propto \exp\left[-\frac{1}{2}\sum_{i=1}^{n}\left(e_{ij} + x_{ij}\beta_j\right)^2 - \frac{1}{2\sigma_\beta^2}\beta_j^2\right] \\
&\propto \exp\left\{-\frac{1}{2\widetilde{\sigma}_{\beta_j}^2}\left(\beta_j - \widetilde{\beta}_j\right)^2\right\},
\end{aligned}
$$

where $e_{ij} = l_i + \sum_{\substack{k=1 \\ k\neq j}}^{p} x_{1k}\beta_k$, $\boldsymbol{l} = (l_1, \ldots, l_n)^{\mathrm{T}}$, $\boldsymbol{x}_j = [x_{1j}, \ldots, x_{nj}]^{\mathrm{T}}$, $\boldsymbol{e}_j = [e_{1j}, \ldots, e_{nj}]^{\mathrm{T}}$, $\widetilde{\sigma}_{\beta_j}^2 = \left(\sigma_\beta^{-2} + \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{x}_j\right)^{-1}$, and $\widetilde{\beta}_j = -\widetilde{\sigma}_{\beta_j}^2\left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{e}_j\right)$. So, the full conditional distribution of β_j is $N\left(\widetilde{\beta}_j, \widetilde{\sigma}_{\beta_j}^2\right)$.

Now, the full conditional distribution of the threshold parameter γ_c is

$$
\begin{aligned}
f(\gamma_c|-) &\propto f(\boldsymbol{y}, \boldsymbol{l}|\boldsymbol{\beta}, \boldsymbol{\gamma}) \\
&\propto \prod_{i=1}^{n} I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i) \\
&\propto \prod_{i\in\{i:y_i=c\}} I_{(\gamma_{c-1}, \gamma_c)}(l_i) \prod_{i\in\{i:y_i=c+1\}} I_{\left(\gamma_c, \gamma_{c+1}\right)}(l_i) \\
&\propto I_{(a_c, b_c)}(\gamma_c),
\end{aligned}
$$

where $a_c = \max\{l_i : y_i = c\}$ and $b_c = \min\{l_i : y_i = c + 1\}$. So $\gamma_c \mid - \sim U(a_c, b_c)$, $c = 1, 2, \ldots, C - 1$. Now, the full conditional distribution of the latent variables is

$$
f(\boldsymbol{l}|-) \propto f(\boldsymbol{y}, \boldsymbol{l}|\boldsymbol{\beta}, \boldsymbol{\gamma}) \propto \prod_{i=1}^{n} f_{L_i}(l_i) I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i)
$$

from which we have that conditional on the rest of parameters, the latent variables are independent truncated normal random variables, that is, $L_i \mid - \sim N\left(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}, 1\right)$ truncated in the interval $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$, $i = 1, \ldots, n$.

The full conditional distribution of σ_β^2 is the same as in the linear regression model described in Chap. 6, so a Gibbs sampler for exploration of the posterior distribution of the parameters of the models can be implemented by iterating the following steps:

1. Initialize the parameters: $\beta_j = \beta_{j0}, j = 1, \ldots, p$, $\gamma_c = \gamma_{c0}, c = 1, \ldots, C - 1$, $l_i = l_{i0}$, $i = 1, \ldots, n$, and $\sigma_\beta^2 = \sigma_{\beta 0}^2$.
2. For each $i = 1, \ldots, n$, simulate l_i from the normal distribution $N\left(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}, 1\right)$ truncated in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$.

3. For each $j = 1, \ldots, p$, simulate from the full conditional distribution of β_j, that is, from a $N\left(\widetilde{\beta}_j, \widetilde{\sigma}^2_{\beta_j}\right)$, where $\widetilde{\sigma}^2_{\beta_j} = \left(\sigma_\beta^{-2} + \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{x}_j\right)^{-1}$ and $\widetilde{\beta}_j = -\widetilde{\sigma}^2_{\beta_j}\left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{e}_j\right)$.
4. For each $c = 1, 2, \ldots, C-1$, simulate from the full conditional distribution of γ_c, $\gamma_c \mid - \sim U(a_c, b_c)$, where $a_c = \max\{l_i : y_i = c\}$ and $b_c = \min\{l_i : y_i = c+1\}$.
5. Simulate σ^2_β from a scaled inverse chi-squared distribution with parameters $\widetilde{\nu}_\beta = \nu_\beta + p$ and $\widetilde{S}_\beta = S_\beta + \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}$, that is, from a $\chi^{-2}\left(\widetilde{\nu}_\beta, \widetilde{S}_\beta\right)$.
6. Repeat 1–5 until a burning period and the desired number of samples are reached.

A similar Gibbs sampler is implemented in the BGLR R package, and the hyperparameters are established in a similar way as the linear regression models described in Chap. 6. When the hyperparameters S_β and ν_β are not specified, by default BGLR is assigned $\nu_\beta = 5$ and to S_β a value such that the prior mode of σ^2_β ($\chi^{-2}(\nu_\beta, S_\beta)$) is equal to half of the "total variance" of the latent variables (Pérez and de los Campos 2014). An implementation of this model can be done with a fixed value of the variance component parameter $\sigma^2_{0\beta}$ by choosing a very large value of the degree of freedom parameter ($\nu_\beta = 10^{10}$) and taking the scale value parameter $S_\beta = \sigma^2_{0\beta}(\nu_\beta + 2)$.

The basic R code with the BGLR implementation of this model is as follows:

```
ETA = list( list( X=X, model='BRR' ) )
A = BGLR(y=y, response_type='ordinal', ETA=ETA, nIter = 1e4, burnIn =
1e3)
Probs = A$Probs
```

where the predictor component is specified in **ETA**: argument **X** specifies the design matrix, whereas the argument model describes the priors of the model's parameters. The response variable vector is given in **y**, and the ordinal model is specified in the response_type argument. In the last line, the posterior means of the probabilities of each category are extracted, and can be used to give an estimation of a particular metric when comparing models or to evaluate the performance of this model, which will be explained later. An application of this ordinal model is given by Montesinos-López et al. (2015a).

Similar to the linear regression model in Chap. 6, when the number of markers to be used in genomic prediction is very large relative to the number of observations, $p \gg n$, the dimension of the posterior distribution (number of parameters in $\boldsymbol{\theta}$, $p + c - 1 + 1$) to be explored can be reduced by simulating the genetic vector effect $\boldsymbol{b} = (b_1, \ldots, b_n)^{\mathrm{T}}$, where $b_i = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}$, $i = 1, \ldots, n$, instead of $\boldsymbol{\beta}$. Because $\boldsymbol{\beta} \mid \sigma^2_\beta \sim N_p\left(\mathbf{0}, \sigma^2_\beta \boldsymbol{I}_p\right)$, the induced distribution of $\boldsymbol{b} = (b_1, \ldots, b_n)^{\mathrm{T}}$ is $\boldsymbol{b} \mid \sigma^2_g \sim N_n\left(\mathbf{0}, \sigma^2_g \boldsymbol{G}\right)$, where $\sigma^2_g = p\sigma^2_\beta$ and $\boldsymbol{G} = \frac{1}{p}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. Then, with this and assuming a scaled inverse chi-squared distribution as prior for σ^2_g, $\sigma^2_g \sim \chi^{-2}\left(\nu_g, S_g\right)$, and a flat prior for the threshold parameters ($\boldsymbol{\gamma}$), an ordinal probit GBLUP Bayesian regression model specification of model (7.1) is given by

$$p_{ic} = P(Y_i = c) = \Phi(\gamma_c + b_i) - \Phi(\gamma_{c-1} + b_i), \quad c = 1, \ldots, C, \tag{7.2}$$

where now $L_i = -b_i + \epsilon_i$ is the latent variable, and Φ is the cumulative normal standard distribution. In matrix form the model for the latent variable can be specified as $\boldsymbol{L} = \boldsymbol{b} + \boldsymbol{\epsilon}$, where $\boldsymbol{L} = (L_1, \ldots, L_n)^{\mathrm{T}}$ and $\boldsymbol{\epsilon} \sim N_n(\boldsymbol{0}, \boldsymbol{I}_n)$. A Gibbs sampler of the posterior of the parameters of this model can be obtained similarly as was done for model (7.1). Indeed, the full conditional density of $\boldsymbol{b}$ is given by

$$\begin{aligned} f(\boldsymbol{b}|-) &\propto f(\boldsymbol{y}, \boldsymbol{l}|\boldsymbol{b}, \boldsymbol{\gamma}) f\left(\boldsymbol{b}|\sigma_g^2\right) \\ &\propto \prod_{i=1}^{n} f_{L_i}(l_i) I_{\left(\gamma_{y_i-1}, \gamma_{y_i}\right)}(l_i) f\left(\boldsymbol{b}|\sigma_g^2\right) \\ &\propto \exp\left[-\frac{1}{2}(\boldsymbol{l}+\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{l}+\boldsymbol{b}) - \frac{1}{2\sigma_g^2}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{G}^{-1}\boldsymbol{b}\right] \\ &\propto \exp\left[-\frac{1}{2}\left(\boldsymbol{b}-\widetilde{\boldsymbol{b}}\right)^{\mathrm{T}}\widetilde{\boldsymbol{\Sigma}}_b\left(\boldsymbol{b}-\widetilde{\boldsymbol{b}}\right)\right], \end{aligned}$$

where $\widetilde{\boldsymbol{\Sigma}}_b = \left(\sigma_g^{-2}\boldsymbol{G}^{-1} + \boldsymbol{I}_n\right)^{-1}$ and $\widetilde{\boldsymbol{b}} = -\widetilde{\boldsymbol{\Sigma}}_b\boldsymbol{l}$. So the full conditional distribution of $\boldsymbol{b}$ is $N_n\left(\widetilde{\boldsymbol{b}}, \widetilde{\boldsymbol{\Sigma}}_b\right)$. Then, a Gibbs sampler exploration of this model can be done by iterating the following steps:

1. Initialize the parameters: $b_i = b_{i0}$, $i = 1, \ldots, n$, $\gamma_c = \gamma_{c0}$, $c = 1, \ldots, C-1$, $l_i = l_{i0}$, $i = 1, \ldots, n$, and $\sigma_\beta^2 = \sigma_{\beta 0}^2$.
2. For each $i = 1, \ldots, n$, simulate l_i from the normal distribution $N\left(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}, 1\right)$ truncated in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$.
3. Simulate $\boldsymbol{b}$ from $N_n\left(\widetilde{\boldsymbol{b}}, \widetilde{\boldsymbol{\Sigma}}_b\right)$, with $\widetilde{\boldsymbol{\Sigma}}_b = \left(\sigma_g^{-2}\boldsymbol{G}^{-1} + \boldsymbol{I}_n\right)^{-1}$ and $\widetilde{\boldsymbol{b}} = -\widetilde{\boldsymbol{\Sigma}}_b\boldsymbol{l}$.
4. For each $c = 1, 2\ldots, C-1$, simulate from the full conditional distribution of γ_c $\gamma_c \mid - \sim U(a_c, b_c)$, where $a_c = \max\{l_i : y_i = c\}$ and $b_c = \min\{l_i : y_i = c+1\}$.
5. Simulate σ_g^2 from a scaled inverse chi-squared distribution with parameters $\widetilde{\nu}_g = \nu_g + n$ and $\widetilde{S}_g = S_g + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}$, that is, from a $\chi^{-2}\left(\widetilde{\nu}_g, \widetilde{S}_g\right)$.
6. Repeat 1–5 until a burning period and the desired number of samples are reached.

This model can also be implemented with the BGLR package:

```
ETA = list( list( K=G, model='RKHS' ) )
A = BGLR(y=y, response_type='ordinal', ETA=ETA, nIter = 1e4, burnIn = 1e3)
Probs = A$Probs
```

where instead of specifying the design matrix and the prior model to the associated regression coefficients, now the genomic relationship matrix $\boldsymbol{G}$ is given, and the corresponding model in this case corresponds to RKHS, which can also be used

when another type of information between the lines is available, for example, the pedigree relationship matrix.

Like the Bayesian linear regression model, other variants of model BRR can be obtained by adopting different prior models for the beta coefficients ($\boldsymbol{\beta}$): FIXED, BayesA, BayesB, BayesC, or BL, for example. See Chap. 6 for details of each of these prior models for the regression coefficients. Indeed, the full conditional distributions used to implement a Gibbs sampler in each of these ordinal models are the same as the corresponding Bayesian linear regression models, except that the full conditional distribution for the variance component of random errors is not needed and two full conditional distributions (for the threshold parameters and the latent variables) are added, where now the latent variable will play the role of the response value in the Bayesian linear regression. These models can also be implemented in the BGLR package.

For example, a Gibbs sampler implementation for ordinal model (7.1) with a BayesA prior (see Chap. 6) can be done by following steps:

1. Initialize the parameters: $\beta_j = \beta_{j0}, j = 1, \ldots, p$, $\gamma_c = \gamma_{c0}$, $c = 1, \ldots, C-1$, $l_i = l_{i0}$, $i = 1, \ldots, n$, and $\sigma^2_\beta = \sigma^2_{\beta 0}$.
2. For each $i = 1, \ldots, n$, simulate l_i from the normal distribution $N\left(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}, 1\right)$ truncated in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$.
3. For each $j = 1, \ldots, p$, simulate from the full conditional distribution of β_j, that is, from a $N\left(\widetilde{\beta}_j, \widetilde{\sigma}^2_{\beta_j}\right)$, where $\widetilde{\sigma}^2_{\beta_j} = \left(\sigma^{-2}_{\beta_j} + \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{x}_j\right)^{-1}$ and $\widetilde{\beta}_j = \widetilde{\sigma}^2_{\beta_j}\left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{e}_j\right)$.
4. For each $c = 1, 2, \ldots, C-1$, simulate from the full conditional distribution of γ_c, $\gamma_c \mid - \sim U(a_c, b_c)$, where $a_c = \max\{l_i : y_i = c\}$ and $b_c = \min\{l_i : y_i = c+1\}$.
5. Simulate from the full conditional of each $\sigma^2_{\beta_j}$

$$\sigma^2_{\beta_j} \mid \boldsymbol{\gamma}, \boldsymbol{l}, \boldsymbol{\beta}_0, \boldsymbol{\sigma}^2_{-j}, S_\beta, \sim \widetilde{\chi}^{-2}_{\widetilde{\nu}_j, \widetilde{S}_{\beta_j}},$$

where $\widetilde{\nu}_{\beta_j} = \nu_\beta + 1$, scale parameter $\widetilde{S}_{\beta_j} = S_\beta + \beta_j^2$, and $\boldsymbol{\sigma}^2_{-j}$ is the vector $\boldsymbol{\sigma}^2_\beta = \left(\sigma^2_{\beta_1}, \ldots, \sigma^2_{\beta_p}\right)$ but without the jth entry.
6. Simulate from the full conditional of S_β

$$\begin{aligned} f(S_\beta|-) &\propto \left[\prod_{j=1}^{p} f\left(\sigma^2_{\beta_j}|S_\beta\right)\right] f(S_\beta) \\ &\propto \prod_{j=1}^{p}\left[\frac{\left(\frac{S_\beta}{2}\right)^{\frac{\nu_\beta}{2}}}{\Gamma\left(\frac{\nu_\beta}{2}\right)\left(\sigma^2_{\beta_j}\right)^{1+\frac{\nu_\beta}{2}}}\exp\left(-\frac{S_\beta}{2\sigma^2_{\beta_j}}\right)\right] S_\beta^{s-1}\exp\left(-rS_\beta\right) \\ &\propto S_\beta^{s+\frac{p\nu_\beta}{2}-1}\exp\left[-\left(r+\frac{1}{2}\sum_{j=1}^{p}\frac{1}{\sigma^2_{\beta j}}\right)S_\beta\right] \end{aligned}$$

which corresponds to the kernel of the gamma distribution with rate parameter $\widetilde{r} = r + \frac{1}{2}\sum_{j=1}^{p} \frac{1}{\sigma^2_{\beta j}}$ and shape parameter $\widetilde{s} = s + \frac{pv_\beta}{2}$, and so, $S_\beta \mid - \sim$ Gamma$(\widetilde{r}, \widetilde{s})$.

7. Repeat steps 2–6 depending on how many values you wish to simulate of the parameter vector $(\boldsymbol{\beta}^{\mathrm{T}}, \boldsymbol{\sigma}^2_\beta, \boldsymbol{\gamma}^{\mathrm{T}})$

The implementation of BayesA for an ordinal response variable in BGLR is as follows:

```
ETA = list( list( X=X, model='BayesA' ) )
A = BGLR(y=y, response_type='ordinal', ETA=ETA, nIter = 1e4, burnIn =
1e3)
Probs = A$Probs
```

The implementation of the other Bayesian ordinal regression models, BayesB, BayesC, and BL, in BGLR is done by only replacing in ETA the corresponding model, as was commented before, see Chap. 6 for details and the difference between all these prior models for the regression coefficients.

7.2.1 Illustrative Examples

Example 1 BRR and GBLUP To illustrate how these models work and how they can be implemented, here we used a toy data set with an ordinal response with five levels and consisting of 50 lines that were planted in three environments (a total of 150 observations) and the genotyped information of 1000 markers for each line. To evaluate the performance of the described models (BRR, BayesA, GBLUP, BayesB, BayesC, and BL), 10 random partitions were used in a cross-validation strategy, where 80% of the complete data set was used to train the model and the rest to evaluate the model in terms of the Brier score (BS) and the proportion of cases correctly classified (PCCC).

Six models were implemented with the BGLR R package, using a burn-in period equal to 1000 and 10,000 iterations, and the default hyperparameter for prior distribution. The results are shown in Table 7.1, where we can appreciate similar performance in terms of the Brier score metric in all models, and a similar but poor performance of all models when the proportion of cases correctly classified (PCCC) was used. With the BRR model only in 4 out of 10 partitions, the values of the PCCC were slightly greater or equal to what can be obtained by random assignation (1/5), while with the GBLUP model this happened in 3 out of 10 partitions. A similar behavior is obtained with the rest of the models. On average, the PCCC values were 0.1566, 0.1466, 0.15, 0.1466, 0.1533, and 0.1433 for the BRR, GBLUP, BayesA, BayesB, BayesC, and Bayes Lasso (BL) models, respectively.

Table 7.1 Brier score (BS) and proportion of cases correctly classified (PCCC) across 10 random partitions, with 80% of the total data set used for training and the rest for testing, under model (7.1) with different priors models to the marker coefficients: ordinal Bayesian Ridge regression model (BRR), BayesA, BayesB, BayesC, and BL (model (7.1)); and under the ordinal GBLUP regression model (7.2)

Model	BRR		GBLUP		BayesA	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.3957	0.1333	0.3937	0.1333	0.3923	0.1333
2	0.3891	0.2000	0.3884	0.2000	0.3896	0.2000
3	0.3977	0.2333	0.3944	0.2333	0.3935	0.2333
4	0.4168	0.1333	0.4154	0.1333	0.4131	0.2000
5	0.4079	0.0667	0.3991	0.0667	0.4013	0.0667
6	0.4068	0.1000	0.4038	0.1000	0.4074	0.1000
7	0.4342	0.1000	0.4258	0.1000	0.4298	0.0667
8	0.3867	0.2000	0.3845	0.1667	0.3858	0.1667
9	0.4145	0.1667	0.4171	0.1000	0.4183	0.1000
10	0.3875	0.2333	0.3862	0.2333	0.3866	0.2333
Average (SD)	0.4036 (0.01)	0.1566 (0.05)	0.4008 (0.01)	0.1466 (0.05)	0.4017 (0.01)	0.15 (0.06)
Model	BayesB		BayesC		BL	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.3924	0.1333	0.3943	0.1333	0.3918	0.1333
2	0.3881	0.2000	0.3885	0.2000	0.3878	0.2000
3	0.3941	0.2667	0.3951	0.2333	0.3935	0.2333
4	0.4141	0.1333	0.4167	0.1333	0.4142	0.1333
5	0.4037	0.0667	0.4044	0.0667	0.4047	0.0667
6	0.3978	0.1000	0.4022	0.1000	0.3984	0.1000
7	0.4268	0.0667	0.4286	0.1000	0.4281	0.0667
8	0.3849	0.1667	0.3862	0.2000	0.3852	0.1667
9	0.4184	0.1000	0.4155	0.1000	0.4169	0.1000
10	0.3869	0.2333	0.3862	0.2667	0.3861	0.2333
Average (SD)	0.4007 (0.01)	0.1466 (0.06)	0.4017 (0.01)	0.1533 (0.06)	0.4006 (0.01)	0.1433 (0.06)

The R code to reproduce the results in Table 7.1 is given in Appendix 1.

In some applications, additional information is available, such as the sites (locations) where the experiments were conducted, environmental covariates, etc., which can be taken into account to improve the prediction performance.

One extension of model (7.1) that takes into account environment effects and environment–marker interaction is given by

$$p_{ic} = P(Y_i = c) = F(\gamma_c - \eta_i) - F(\gamma_{c-1} - \eta_i), c = 1, \ldots, C, \tag{7.3}$$

where now the predictor $\eta_i = \boldsymbol{x}_{Ei}^{\mathrm{T}}\boldsymbol{\beta}_E + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \boldsymbol{x}_{EMi}^{T}\boldsymbol{\beta}_{EM}$, in addition to the marker effects ($\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}$) in the second summand, is included the environment and the

environment–marker interaction effects, respectively. In the latent variable representation, this model is equivalently expressed as

$$P(Y_i = c) = P(\gamma_{c-1} \leq L_i \leq \gamma_c), c = 1, \ldots, C,$$

where $\boldsymbol{L} = (L_1, \ldots, L_n)^{\mathrm{T}} = \mathbf{X}_E\boldsymbol{\beta}_E + \boldsymbol{X\beta} + \boldsymbol{X}_{EM}\boldsymbol{\beta}_{EM} + \boldsymbol{\epsilon}$ is the vector with latent random variables of all observations, $\boldsymbol{\epsilon} \sim N_n(\mathbf{0}, \boldsymbol{I}_n)$ is a random error vector, $\boldsymbol{X}_E$, $\boldsymbol{X}$, and $\boldsymbol{X}_{EM}$ are the design matrices of the environments, markers, and environment–marker interactions, respectively, while $\boldsymbol{\beta}_E$ and $\boldsymbol{\beta}_{EM}$ are the vectors of the environment effects and the interaction effects, respectively, with a prior distribution that can be specified as was done for $\boldsymbol{\beta}$. In fact, with the BGLR function, it is also possible to implement all these extensions, since it allows using any of the several priors included here: FIXED, BRR, BayesA, BayesB, BayesC, and BL. For example, the basic BGLR code to implement model (7.3) with a flat prior ("FIXED") for the environment effects, a BRR prior for marker effects and for the environment–marker interaction effects, is as follows:

```
X = scale(X)
#Environment matrix design
XE = model.matrix(~Env,data=dat_F)[,-1]
#Environment-marker interaction matrix design
XEM = model.matrix(~0+X:Env,data=dat_F)
ETA = list(list(X=XE, model='FIXED'), list(X=X, model='BRR'),
          list(X=XEM, model='BRR'))#Predictor
A = BGLR(y=y,response_type='ordinal',ETA=ETA, nIter = 1e4,burnIn =
1e3,verbose = FALSE)
Probs = A$probs
```

where **dat_F** is the data file that contains all the information of how the data was collected (GID: Lines or individuals; Env: Environment; y: response variable of the trait). Other desired prior models to beta coefficients of each predictor component are obtained only by replacing the "model" argument of each of the three components of the predictor. For example, for a BayesA prior model for the marker effects, in the second sub-list we must use model='**BayesA**'.

The latent random vector of model (7.1) under the GBLUP specification, plus genotypic and environment×genotypic interaction effects, takes the form

$$\boldsymbol{L} = \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{Z}_L\boldsymbol{g} + \boldsymbol{Z}_{LE}\boldsymbol{gE} + \boldsymbol{\epsilon} \tag{7.4}$$

which is like model (6.7) in Chap. 6, where $\boldsymbol{Z}_L$ and $\boldsymbol{Z}_{LE}$ are the incident matrices of the genotypes and environment×genotype interaction effects, respectively, and $\boldsymbol{g}$ and $\boldsymbol{gE}$ are the corresponding random effects which have distributions $N_J\left(\boldsymbol{O}, \sigma_g^2\boldsymbol{G}\right)$ and $N_J\left(\boldsymbol{O}, \sigma_{gE}^2(\boldsymbol{I}_I \otimes \boldsymbol{G})\right)$, respectively, where J is the number of different lines in the data set. This model can be trained with the BGLR package as follows:

```
I = length(unique(dat_F$Env))
XE = model.matrix(~0+Env,data=dat_F)[,-1]
Z_L = model.matrix(~0+GID,data=dat_F,xlev = list(GID=unique
(dat_F$GID)))
K_L = Z_L %*%G%*%t(Z_L)
Z_LE = model.matrix(~0+GID:Env,data=dat_F,
          xlev = list(GID=unique(dat_F$GID),Env = unique(dat_F$Env)))
K_LE = Z_LE%*%kronecker(diag(I),G)%*%t(Z_LE)
ETA = list( list(model='FIXED',X=XE),
                  list( model = 'RHKS', K = K_L )),
                  list(model='RKHS',K=K_LE )
A   = BGLR(y, response_type = "ordinal", ETA = ETA, nIter = 1e4, burnIn =
1e3)
Probs = A$Probs
```

where **dat_F** is as before, that is, the data file that contains all the information of how the data were collected. Similar specification can be done when other kinds of covariates are available, for example, environmental covariates.

Example 2 Environments + Markers + Marker×Environment Interaction This example illustrates how models (7.3) and (7.4) can be fitted and used for prediction. Here we used the same data set as in Example 1, but now the environment and environment×marker effects are included in the predictor. For model (7.3), to the environment effect a flat prior is assigned [FIXED, a normal distribution with mean 0 and very large variance (10^{10})], and one of BRR, BayesA, BayesB, BayesC, or BL prior model is assigned to the marker and marker×environment interaction effects. For model (7.4), a flat prior is assigned for environment effects. The performance evaluation was done using the same 10 random partitions used in Example 1, where 80% of the complete data set was used for training the model and the rest for testing, in which the Brier score (BS) and the proportion of correctly classified (PCCC) metrics were computed.

The results are shown in Table 7.2. They indicate an improved prediction performance of these models compared to the models fitted in Example 1 which only takes into account the marker effects (see Table 7.1). However, this improvement is only slightly better under the Brier score, because the reduction in the average BS across all 10 partitions was 1.55, 3.74, 0.96, 0.83, 1.41, and 1.32% for the BRR, GBLUP, BayesA, BayesB, BayesC, and BL, respectively. A more notable improvement was obtained with the PCCC criteria, where now for all models in about 8 out of the 10 partitions, the value of this metric was greater than the one obtained by random chance only. Indeed, the average values across the 10 partitions were 0.27, 0.28, 0.28, 0.27, 0.28, and 0.27, for the BRR, GBLUP, BayesA, BayesB, BayesC, and BL, respectively. This indicates 74.47, 90.90, 86.67, 86.36, 80.43, and 86.05% improvement for these models with respect to their counterpart models when only marker effects or genomic effects were considered. So, greater improvement was observed with the GBLUP model with both metrics, but finally the performance of all models is almost undistinguishable but with an advantage in time execution of the GBLUP model with respect to the rest. These issues were

Table 7.2 Brier score (BS) and proportion of cases correctly classified (PCCC) across 10 random partitions, with 80% of the total data set used for training and the rest for testing, under model (7.3) with different prior models for the marker effects and the environment×marker interaction: BRR, BayesA, BayesB, BayesC, and BL; and under the ordinal GBLUP regression model (7.4)

Model	BRR		GBLUP		BayesA	
PT	BS	PCC	BS	PCC	BS	PCC
1	0.3942	0.3333	0.3721	0.3667	0.3925	0.2333
2	0.3872	0.3000	0.3881	0.2667	0.3871	0.3000
3	0.4081	0.2667	0.4019	0.2667	0.4054	0.2667
4	0.4428	0.1333	0.4375	0.0667	0.4448	0.1667
5	0.3853	0.2667	0.3602	0.3000	0.3837	0.2333
6	0.3905	0.3667	0.3681	0.3000	0.3899	0.3667
7	0.3995	0.1333	0.3849	0.2000	0.4007	0.1667
8	0.3737	0.3000	0.3690	0.3333	0.3816	0.3333
9	0.3999	0.3000	0.3893	0.3333	0.3989	0.3333
10	0.3937	0.3333	0.3925	0.3667	0.3953	0.4000
Average (SD)	0.3975 (0.01)	0.2733 (0.07)	0.3863 (0.02)	0.28 (0.09)	0.3979 (0.01)	0.28 (0.08)
Model	BayesB		BayesC		BL	
PT	BS	PCC	BS	PCC	BS	PCC
1	0.3913	0.3000	0.3894	0.2667	0.3890	0.2333
2	0.3868	0.3333	0.3868	0.3000	0.3851	0.3333
3	0.4077	0.2667	0.4081	0.2667	0.4063	0.2667
4	0.4456	0.1333	0.4420	0.1667	0.4403	0.1667
5	0.3809	0.2667	0.3805	0.2667	0.3814	0.2667
6	0.3883	0.3667	0.3862	0.3667	0.3846	0.3667
7	0.4026	0.1333	0.3992	0.1333	0.3964	0.1333
8	0.3735	0.3000	0.3735	0.3000	0.3739	0.3000
9	0.4026	0.3333	0.4015	0.3333	0.4037	0.3000
10	0.3948	0.3000	0.3948	0.3667	0.3939	0.3000
Average (SD)	0.3974 (0.01)	0.2733 (0.07)	0.3961 (0.01)	0.2766 (0.07)	0.3954 (0.01)	0.2666 (0.07)

pointed out in Chaps. 5 and 6. This difference is even greater when the number of markers is larger than the number of observations.

Example 3 Binary Traits For this example, we used the EYT Toy data set consisting of 40 lines, four environments (Bed5IR, EHT, Flat5IR, and LHT), and a response binary variable based on plant Height (0 = low, 1 = high). For this example, marker information is not available, only the genomic relationship matrix for the 40 lines. So, only the models (M3 and M4) in (7.3) and (7.4) are fitted, and the performance prediction for these models was evaluated using cross-validation. Also, in this comparison model, (M5) (7.5) is added but without the line×environment interaction effects, that is, only the environment effect and the genetic effects are taken into account in the linear predictor:

Table 7.3 Brier score (BS) and proportion of cases correctly classified (PCCC) across 10 random partitions, with 80% of the total data set used for training and the rest for testing, under models (7.3), (7.4), and (7.5) with a flat prior for environment effects

Model	M3		M4		M5	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.1841	0.8438	0.2250	0.7500	0.2166	0.7813
2	0.1580	0.8438	0.2307	0.5938	0.2221	0.7188
3	0.1960	0.7813	0.2357	0.6563	0.2308	0.6563
4	0.2261	0.7188	0.2119	0.6875	0.2102	0.6563
5	0.2108	0.6250	0.3275	0.4063	0.3174	0.4063
6	0.2070	0.7188	0.2280	0.6563	0.2242	0.6250
7	0.2094	0.6563	0.2261	0.6250	0.2248	0.5938
8	0.1864	0.7500	0.2322	0.7188	0.2293	0.6875
9	0.1813	0.7813	0.2643	0.5000	0.2561	0.5313
10	0.2310	0.6563	0.2865	0.5625	0.2878	0.5313
Average (SD)	0.199 (0.02)	0.7375 (0.07)	0.2468 (0.03)	0.6156 (0.1)	0.2419 (0.03)	0.6187 (0.1)

$$\boldsymbol{L} = \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{Z}_L\boldsymbol{g} + \boldsymbol{\epsilon} \tag{7.5}$$

The results are presented in Table 7.3 with the BS and PCCC metrics obtained in each partition of the random CV strategy. From this we can appreciate that the best performance with both metrics was obtained with the model that considered only the genetic effects (M3; (7.3)). On average, models M4 and M5 gave a performance in terms of BS, that was 24.02 and 19.79% greater than the corresponding performance of model M3, while with model M3, on average across the 10 partitions, an improvement of 21.57 and 19.19% in terms of PCCC was obtained with regard to models M4 and M5. The difference between these last two models is only slight; the average BS value of the first model was 2.01% greater than that of the second, and the PCCC of the second model was 0.50% greater than that of the first. The R code to reproduce the results is in Appendix 3.

7.3 Ordinal Logistic Regression

As described at the beginning of this chapter, the ordinal logistic model is given in model (7.1) but with F the cumulative logistic distribution. Again, as in the ordinal probit regression model, the posterior distribution of the parameter is not easy to simulate and numerical methods are required. Here we describe the Gibbs sampler proposed by Montesinos-López et al. (2015b), which in addition to the latent variable L_i in the representation of model (7.1), the parameter vectors are also augmented with a Pólya-Gamma latent random variable.

By using the following identity proposed by Polson et al. (2013):

$$\frac{[\exp(\eta)]^a}{[1+\exp(\eta)]^b} = 2^{-b}\exp(k\eta)\int_0^\infty \exp\left(-\frac{\eta^2}{2}\omega\right) f_\omega(\omega; b, 0)d\omega,$$

where $k = a - b/2$ and $f_\omega(\omega; b, 0)$ is the density of a Pólya-Gamma random variable ω with parameters b and $d = 0$ ($\omega \sim \text{PG}(b, d)$, (see Polson et al. (2013) for details), the joint distribution of the vector of observations, $\boldsymbol{Y} = (Y_1, \ldots, Y_n)^{\text{T}}$, and the latent variables $\boldsymbol{L} = (L_1, \ldots, L_n)^{\text{T}}$ can be expressed as

$$\begin{aligned} f(\boldsymbol{y}, \boldsymbol{l}|\boldsymbol{\beta}, \boldsymbol{\gamma}) &= \prod_{i=1}^n f_{L_i}(l_i) I_{(\gamma_{y_i-1}, \gamma_{y_i})}(l_i) \\ &= \prod_{i=1}^n \frac{\exp(-l_i - \eta_i)}{[1+\exp(-l_i-\eta_i)]^2} I_{(\gamma_{y_i-1}, \gamma_{y_i})}(l_i) \\ &\propto \prod_{i=1}^n \left[\int_0^\infty \exp\left(-\frac{(l_i+\eta_i)^2}{2}\omega_i\right) f_{\omega_i}(\omega_i; 2, 0)d\omega_i\right] I_{(\gamma_{y_i-1}, \gamma_{y_i})}(l_i), \end{aligned}$$

where $\eta_i = \boldsymbol{x}_i^{\text{T}}\boldsymbol{\beta}$ and $\omega_1, \ldots, \omega_n$ are independent random variables with the same Pólya-Gamma distribution with parameters $b = 2$ and $d = 0$.

Now, the Bayesian specification developed in Montesinos-López et al. (2015b) assumes the same priors as for the ordinal probit model (BRR prior model), except that now for the threshold parameter vector $\boldsymbol{\gamma}$, the prior distribution proposed by Sorensen et al. (1995) is adopted, which is the distribution of the order statistics of a random sample of size $C - 1$ of the uniform distribution in (γ_L, γ_U), that is,

$$f(\boldsymbol{\gamma}) = (C-1)!\left(\frac{1}{\gamma_U - \gamma_L}\right)^{C-1} I_{\{\boldsymbol{\gamma}\in S_\gamma\}},$$

where $\boldsymbol{S}_\gamma = \{(\gamma_1, \ldots, \gamma_{C-1}) : \gamma_L < \gamma_1 < \gamma_2 < \cdots < \gamma_{C-1} < \gamma_U\}$.

Then, by conditioning on the rest of parameters, including the latent Pólya-Gamma random variables $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n)^{\text{T}}$, the full conditional of γ_c is given by

$$\begin{aligned} f(\gamma_c|-) &\propto f(\boldsymbol{y}, \boldsymbol{l}, \boldsymbol{\omega}|\boldsymbol{\beta}, \boldsymbol{\gamma}) f(\boldsymbol{\gamma}) \\ &\propto \left[\prod_{i=1}^n I_{(\gamma_{y_i-1}, \gamma_{y_i})}(l_i)\right](C-1)!\left(\frac{1}{\gamma_U - \gamma_L}\right)^{C-1} I_{S_\gamma}(\boldsymbol{\gamma}) \\ &\propto I_{(a_c, b_c)}(\gamma_c) \end{aligned}$$

where $f(\boldsymbol{y}, \boldsymbol{l}, \boldsymbol{\omega}|\boldsymbol{\beta}, \boldsymbol{\gamma})$ is the joint density of the observations and the vector of the latent random variables, $\boldsymbol{L}$ and $\boldsymbol{\omega}$, $a_c = \max\{\gamma_{c-1}, \max\{l_i : y_i = c\}\}$, and $b_c = \min\{\gamma_{c+1}, \min\{l_i : y_i = c+1\}\}$. So the full conditional for the threshold parameters is also uniform, $\gamma_c \mid - \sim U(a_c, b_c)$, $c = 1, 2\ldots, C-1$.

Now, the conditional distribution of β_j (jth element of $\boldsymbol{\beta}$) is given by

$$\begin{aligned} f\left(\beta_j|\sigma_\beta^2, \boldsymbol{\gamma}, \boldsymbol{L}, \boldsymbol{\omega}\right) &\propto f(\boldsymbol{y}, \boldsymbol{l}, \boldsymbol{\omega}|\boldsymbol{\beta}, \boldsymbol{\gamma}) f\left(\boldsymbol{\beta}|\sigma_\beta^2\right) \\ &\propto \exp\left[-\frac{1}{2}\sum_i^n \omega_i \frac{(l_i+\eta_i)^2}{2} - \frac{1}{2\sigma_\beta^2}\beta_j^2\right] \\ &\propto \exp\left[-\frac{1}{2}\sum_i^n \omega_i \frac{\left(e_{ij}+x_{ij}\beta_j\right)^2}{2} - \frac{1}{2\sigma_\beta^2}\beta_j^2\right] \\ &\propto \exp\left\{-\frac{1}{2\widetilde{\sigma}_{\beta_j}^2}\left(\beta_j - \widetilde{\beta}_j\right)^2\right\}, \end{aligned}$$

where $e_{ij} = l_i + \sum_{\substack{k=1 \\ k\neq j}}^p x_{1k}\beta_k$, $\boldsymbol{e}_j = [e_{1j}, \ldots, e_{nj}]^{\mathrm{T}}$, $\boldsymbol{x}_j = [x_{1j}, \ldots, x_{nj}]^{\mathrm{T}}$, $\widetilde{\sigma}_{\beta_j}^2 = \left(\sigma_\beta^{-2} + \sum_{i=1}^n \omega_i x_{ij}^2\right)^{-1}$, and $\widetilde{\beta}_j = -\widetilde{\sigma}_{\beta_j}^2\left(\sum_{i=1}^n w_i x_{ij} e_{ij}\right)$. From this, the full conditional distribution of β_j is a normal distribution with mean $\widetilde{\beta}_j$ and variance $\widetilde{\sigma}_{\beta_j}^2$.

The full conditional distribution of σ_β^2 is the same as the one obtained for the ordinal probit model, $\sigma_\beta^2 \mid - \sim \chi^{-2}\left(\widetilde{\nu}_\beta, \widetilde{S}_\beta\right)$ with $\widetilde{\nu}_\beta = \nu_\beta + p$ and $\widetilde{S}_\beta = S_\beta + \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}$. In a similar fashion, for the latent variable L_i, $i = 1, \ldots, n$, it can be seen that its full conditional distributions are also truncated normal in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$ but with mean parameter $-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}$ and variance $1/\omega_i$, i.e., $L_i \mid - \sim N\left(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}, \omega_i^{-1}\right)$ truncated in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$, for each $i = 1, \ldots, n$. Finally, by following Eq. (5) in Polson et al. (2013), note that the full joint conditional distribution of the Pólya random variables $\boldsymbol{\omega}$ can be written as

$$\begin{aligned} f(\boldsymbol{\omega}|-) &\propto f(\boldsymbol{y}, \boldsymbol{l}, \boldsymbol{\omega}|\boldsymbol{\beta}, \boldsymbol{\gamma}) \\ &\propto \prod_{i=1}^n \exp\left(-\frac{(l_i+\eta_i)^2}{2}\omega_i\right) f_{\omega_i}(\omega_i; 2, 0) \\ &\propto \prod_{i=1}^n f_{\omega_i}(\omega_i; 2, l_i+\eta_i). \end{aligned}$$

From here, conditionally to $\boldsymbol{\beta}, \sigma_\beta^2, \boldsymbol{\gamma}$, and $\boldsymbol{L}$, $\omega_1, \ldots, \omega_n$ are independently Pólya-Gamma random variables with parameters $b = 2$ and $d = l_i + \eta_i$, $i = 1, \ldots, n$, respectively, that is, $\omega_i \mid - \sim \mathrm{PG}(2, l_i + \eta_i)$, $i = 1, \ldots, n$.

With the above derived full conditionals, a Gibbs sampler exploration of this ordinal logistic regression model can be done with the following steps:

1. Initialize the parameters: $\beta_j = \beta_{j0}, j = 1, \ldots, p$, $\gamma_c = \gamma_{c0}, c = 1, \ldots, C-1$, $l_i = l_{i0}$, $i = 1, \ldots, n$, and $\sigma_\beta^2 = \sigma_{\beta 0}^2$.
2. Simulate $\omega_1, \ldots, \omega_n$ independently Pólya-Gamma random variables with parameters $b = 2$ and $d = l_i + \eta_i$, $i = 1, \ldots, n$.
3. For each $i = 1, \ldots, n$, simulate l_i from the normal distribution $N\left(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}, \omega_i^{-1}\right)$ truncated in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$.
4. For each $j = 1, \ldots, p$, simulate from the full conditional distribution of β_j, that is, from a $N\left(\widetilde{\beta}_j, \widetilde{\sigma}_{\beta_j}^2\right)$, where $\widetilde{\sigma}_{\beta_j}^2 = \left(\sigma_\beta^{-2} + \sum_{i=1}^n \omega_i x_{ij}^2\right)^{-1}$ and $\widetilde{\beta}_j = -\widetilde{\sigma}_{\beta_j}^2\left(\sum_{i=1}^n \omega_i x_{ij} e_{ij}\right)$.
5. For each $c = 1, 2\ldots, C-1$, simulate from the full conditional distribution of γ_c, $\gamma_c \mid - \sim U(a_c, b_c)$, where $a_c = \max\{\gamma_{c-1}, \max\{l_i : y_i = c\}\}$ and $b_c = \min\{\gamma_{c+1}, \min\{l_i : y_i = c+1\}\}$.
6. Simulate σ_β^2 from a scaled inverse chi-squared distribution with parameters $\widetilde{\nu}_\beta = \nu_\beta + p$ and $\widetilde{S}_\beta = S_\beta + \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}$, that is, from a $\chi^{-2}\left(\widetilde{\nu}_\beta, \widetilde{S}_\beta\right)$.
7. Repeat 1–6 until a burning period and a desired number of samples are reached.

Similar modifications can be done to obtain Gibbs samplers corresponding to other prior adopted models for the beta coefficients (FIXED, BayesA, BayesB, BayesC, or BL; see Chap. 6 for details of these priors). Also, for the ordinal logistic GBLUP Bayesian regression model specification as done in (7.2) for the ordinal probit model, a Gibbs sampler implementation can be done following the steps below, which can be obtained directly from the ones described above:

1. Initialize the parameters: $b_i = b_{i0}, j = 1, \ldots, n$, $\gamma_c = \gamma_{c0}, c = 1, \ldots, C-1$, $l_i = l_{i0}$, $i = 1, \ldots, n$, and $\sigma_g^2 = \sigma_{g0}^2$.
2. Simulate $\omega_1, \ldots, \omega_n$ independently Pólya-Gamma random variables with parameters $b = 2$ and $d = l_i + b_i$, $i = 1, \ldots, n$.
3. For each $i = 1, \ldots, n$, simulate l_i from the normal distribution $N\left(-b_i, \omega_i^{-1}\right)$ truncated in $\left(\gamma_{y_i-1}, \gamma_{y_i}\right)$.
4. Simulate $\boldsymbol{b}$ from $N_n\left(\widetilde{\boldsymbol{b}}, \widetilde{\boldsymbol{\Sigma}}_b\right)$, with $\widetilde{\boldsymbol{\Sigma}}_b = \left(\sigma_g^{-2}\boldsymbol{G}^{-1} + \boldsymbol{D}_\omega\right)^{-1}$ and $\widetilde{\boldsymbol{b}} = -\widetilde{\boldsymbol{\Sigma}}_b\boldsymbol{D}_\omega\boldsymbol{l}$, where $\boldsymbol{D}_\omega = \mathrm{Diag}(\omega_1, \ldots, \omega_n)$.
5. For each $c = 1, 2\ldots, C-1$, simulate from the full conditional distribution of γ_c, $\gamma_c \mid - \sim U(a_c, b_c)$, where $a_c = \max\{\gamma_{c-1}, \max\{l_i : y_i = c\}\}$ and $b_c = \min\{\gamma_{c+1}, \min\{l_i : y_i = c+1\}\}$.
6. Simulate σ_g^2 from a scaled inverse chi-squared distribution with parameters $\widetilde{\nu}_g = \nu_g + n$ and $\widetilde{S}_g = S_g + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}$, that is, from a $\chi^{-2}\left(\widetilde{\nu}_g, \widetilde{S}_g\right)$.
7. Repeat 1–6 until a burning period and the desired number of samples are reached.

There is a lot of empirical evidence that there are no large differences between the prediction performance of the probit or logistic regression models. For this reason, here we only explained the Gibbs sampler for ordinal data under a logistic framework and did not provide illustrated examples. Also, we did not provide illustrative examples because it is not possible to implement this logistic ordinal model in BGLR.

7.4 Penalized Multinomial Logistic Regression

An extension of the logistic regression model described in Chap. 3 is the multinomial regression model that is also used to explain or predict a categorical nominal response variable (that does not have any natural order) with more than two categories. For example, a study could investigate the association of markers with diabesity (diabetes and obesity, diabetes but no obesity, obesity but no diabetes, and no diabetes and no obesity); another example could be to study the effects of age group on the histological subtypes of woman cancer (adenocarcinoma, adenosquamous, others), predict the preference in an election among four candidates (C1, C2, C3, and C4) using socioeconomic and demographic variables, etc.

The multinomial logistic regression model assumes that, conditionally to a covariate $\boldsymbol{x}_i$, a multinomial response random variable Y_i takes one of the categories 1, 2, ..., C, with the following probabilities:

$$P(Y_i = c|\boldsymbol{x}_i) = \frac{\exp\left(\beta_{0c} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_c\right)}{\sum_{l=1}^{C}\exp\left(\beta_{0l} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_l\right)}, c = 1, \ldots, C, \tag{7.6}$$

where $\boldsymbol{\beta}_c$, $c = 1, \ldots, C$, is a vector of coefficients of the same dimension as $\boldsymbol{x}$.

Model (7.6) is not identifiable because by evaluating these probabilities with the parameter values $\left(\beta_{0c}^{*}, \boldsymbol{\beta}_c^{*\mathrm{T}}\right) = \left(\beta_{0c} + \beta_{0c}^{**}, \boldsymbol{\beta}_{0c}^{\mathrm{T}} + \boldsymbol{\beta}_c^{**\mathrm{T}}\right)$, $c = 1, \ldots, C$, where β_{0c}^{**} and $\boldsymbol{\beta}_c^{**}$ are arbitrary constants and vectors, give equal probabilities than when computing this with the original parameter values $\left(\beta_{0c}, \boldsymbol{\beta}_c^{\mathrm{T}}\right)$, $c = 1, \ldots, C$. A common constraint that avoids this lack of identifiability is to set $\left(\beta_{0C}, \boldsymbol{\beta}_C^{\mathrm{T}}\right) = \left(0, \mathbf{0}^{\mathrm{T}}\right)$, although any one of the other $C - 1$ of the vectors could be chosen.

With such a constraint ($\left(\beta_{0C}, \boldsymbol{\beta}_C^{\mathrm{T}}\right) = \left(0, \mathbf{0}^{\mathrm{T}}\right)$), we can identify a model by assuming that the effects of the covariate vector $\boldsymbol{x}_i$ over the log "odds" of each of the categories $c = 1, \ldots, C - 1$ with respect to category C (baseline category) are given (Agresti 2012) by

$$\log\left(\frac{P(Y_i = c|x)}{P(Y_i = C|x)}\right) = \beta_{0c} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_c, \quad c = 1, \ldots, C - 1,$$

where the effects of $\boldsymbol{x}_i$ depend on the chosen response baseline category. Similar expressions can be obtained when using the unconstrained model in (7.6). From

these relationships, the log odds corresponding to the probabilities of any two categories, c and l, can be derived:

$$\log\left(\frac{P(Y_i = c|x)}{P(Y_i = l|x)}\right) = \beta_{0c} - \beta_{0l} + \boldsymbol{x}_i^{\mathrm{T}}(\boldsymbol{\beta}_c - \boldsymbol{\beta}_l), \quad c = 1, \ldots, C-1.$$

Sometimes the number of covariates is larger than the number of observations (for example, in expression arrays and genomic prediction where the number of markers is often larger than the number of phenotyped individuals), so the conventional constraint described above to force identifiability in the model is not enough and some identifiability problems remain. One way to avoid this problem is to use similar quadratic regularization maximum likelihood estimation methods (Zhu and Hastie 2004) as done for some models in Chap. 3, but here only on the slopes ($\boldsymbol{\beta}_c$) for the covariates, under which setting the constraints mentioned before are no longer necessary.

For a given value of the regularization parameter, $\lambda > 0$, the regularized maximum likelihood estimation of the beta coefficients, $\boldsymbol{\beta} = \left(\beta_{0c}, \boldsymbol{\beta}_c^{\mathrm{T}}, \beta_{02}, \boldsymbol{\beta}_2^{\mathrm{T}}, \ldots, \beta_{0C}, \boldsymbol{\beta}_C^{\mathrm{T}}\right)^{\mathrm{T}}$, is the value of this that maximizes the penalized log-likelihood:

$$\ell_p(\boldsymbol{\beta}; \boldsymbol{y}) = \ell(\boldsymbol{\beta}; \boldsymbol{y}) - \lambda \sum_{c=1}^{C} \boldsymbol{\beta}_c^{\mathrm{T}} \boldsymbol{\beta}_c, \tag{7.7}$$

where $\ell(\boldsymbol{\beta}; \boldsymbol{y}) = \log\left[L(\boldsymbol{\beta}; \boldsymbol{y})\right]$ is the logarithm of the likelihood and takes the form:

$$\begin{aligned}\ell(\boldsymbol{\beta}; \boldsymbol{y}) &= \sum_{i=1}^{n} \log\left[P(Y_i = y_i|\boldsymbol{x}_i)\right] = \sum_{i=1}^{n} \sum_{c=1}^{C} I_{\{y_i=c\}} \log\left[\frac{\exp\left(\beta_{0c} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_c\right)}{\sum_{l=1}^{C} \exp\left(\beta_{0l} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_l\right)}\right] \\ &= \sum_{i=1}^{n} \sum_{c=1}^{C} I_{\{y_i=c\}} \left(\beta_{0c} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_c\right) - \sum_{i=1}^{n} \log\left[\sum_{l=1}^{C} \exp\left(\beta_{0l} + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_l\right)\right]\end{aligned} \tag{7.8}$$

When p is large ($p \gg n$), direct optimization of $\ell_p(\boldsymbol{\beta}; \boldsymbol{y})$ is almost impossible. An alternative is to use the sequential minimization optimization algorithm proposed by Zhu and Hastie (2004), which is applied after a transformation trick is used to make the involved computations feasible, because the number of parameters in the optimization is reduced to only $(n + 1)C$ instead of $(p + 1)C$.

Another alternative available in the glmnet package is the one proposed by Friedman et al. (2010) that is similar to that of the logistic Ridge regression in Chap. 3. This consists of maximizing (7.7) by using a block-coordinate descent strategy, where each block is formed by the beta coefficients corresponding to each class, $\boldsymbol{\beta}_c^{*\mathrm{T}} = \left(\beta_{0c}, \boldsymbol{\beta}_c^{\mathrm{T}}\right)$, but with $\ell(\boldsymbol{\beta}; \boldsymbol{y})$ replaced by a quadratic approximation with respect to beta coefficients of the chosen block, $\left(\beta_{0c}, \boldsymbol{\beta}_c^{\mathrm{T}}\right)$, at the current values

of $\boldsymbol{\beta}$ ($\widetilde{\boldsymbol{\beta}}$). That is, the update of block c is achieved by maximizing the following function with respect to β_{0c} and $\boldsymbol{\beta}_c$:

$$f_c(\beta_{0c}, \boldsymbol{\beta}_c) = \ell_c^*(\boldsymbol{\beta}; \boldsymbol{y}) - \lambda \boldsymbol{\beta}_c^{\mathrm{T}} \boldsymbol{\beta}_c, \tag{7.9}$$

where $\ell_c^*(\boldsymbol{\beta}; \boldsymbol{y}) = -\frac{1}{2} \sum_{i=1}^n w_{ic} \left(y_{ic}^* - \beta_{0c} - \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{\beta}_c \right)^2 + \widetilde{c}$ is the second-order Taylor approximation of $\ell(\boldsymbol{\beta}; \boldsymbol{y})$ with respect to the beta coefficients that conform block c, $\left(\beta_{0c}, \boldsymbol{\beta}_c^{\mathrm{T}}\right)$, about the current estimates $\widetilde{\boldsymbol{\beta}} = \left(\widetilde{\beta}_{0c}, \widetilde{\boldsymbol{\beta}}_c^{\mathrm{T}}, \widetilde{\beta}_{02}, \widetilde{\boldsymbol{\beta}}_2^{\mathrm{T}}, \ldots, \widetilde{\beta}_{0C}, \widetilde{\boldsymbol{\beta}}_C^{\mathrm{T}}\right)^{\mathrm{T}}$; $y_{ic}^* = \widetilde{\beta}_{0c} + \boldsymbol{x}_i^{\mathrm{T}} \widetilde{\boldsymbol{\beta}}_c + w_{ic}^{-1} \left(I_{\{y_i = c\}} - \widetilde{p}_c(\boldsymbol{x}_i) \right)$ is the "working response," $w_{ic} = \widetilde{p}_c(\boldsymbol{x}_i) \times (1 - \widetilde{p}_c(\boldsymbol{x}_i))$, and $\widetilde{p}_c(\boldsymbol{x}_i)$ is $P(Y_i = c | \boldsymbol{x}_i)$ given in (7.6) but evaluated at the current parameter values.

When $\boldsymbol{X}$ is the design matrix with standardized independent variables, the updated parameters of block c, $\widehat{\boldsymbol{\beta}}_c^{*\mathrm{T}}$, can be obtained by the following formula:

$$\widehat{\boldsymbol{\beta}}_c^{*\mathrm{T}} = \left(\boldsymbol{X}^{*\mathrm{T}} \boldsymbol{W}_c \boldsymbol{X}^* + \lambda \boldsymbol{D} \right)^{-1} \boldsymbol{X}^{*\mathrm{T}} \boldsymbol{W}_c \boldsymbol{y}^*,$$

where $\boldsymbol{X}^* = [\boldsymbol{1}_n \, \boldsymbol{X}]$, $\boldsymbol{W}_c = \mathrm{Diag}(w_{1c}, \ldots, w_{nc})$, $\boldsymbol{y}^* = \left(y_1^*, \ldots, y_n^*\right)^{\mathrm{T}}$, and $\boldsymbol{D}$ is an identity matrix of dimension $(p + 1) \times (p + 1)$ except that in the first entry we have the value of 0 instead of 1. However, in the context of $p \gg n$, a non-prohibited optimization of (7.9) is achieved by using coordinate descent methods as done in the glmnet package and commented in Chap. 3.6.2.

For other penalization terms, a very similar algorithm to the one described before can be used. For example, for Lasso penalty, the penalized likelihood (7.7) is modified as

$$\ell_p(\boldsymbol{\beta}; \boldsymbol{y}) = \ell(\boldsymbol{\beta}; \boldsymbol{y}) - \lambda \sum_{c=1}^{C} \sum_{j=1}^{p} |\beta_{cj}| \tag{7.10}$$

and block updating can be done as in (7.9), except that now $\boldsymbol{\beta}_c^{\mathrm{T}} \boldsymbol{\beta}_c$ is replaced by $\sum_{j=1}^p |\beta_{cj}|$. Like the penalized logistic regression studied in Chap. 3, the more common approach for choosing the "optimal" regularization parameter λ in the penalized multinomial regression model in (7.7) is by using a k-fold cross-validation strategy with misclassification error as metrics. This will be used here. For more details, see Friedman et al. (2010). It is important to point out that the tuning parameter λ used in glmnet is equal to the one used in the penalized log-likelihood (7.7) and (7.10) but divided by the number of observations.

7.4.1 *Illustrative Examples for Multinomial Penalized Logistic Regression*

Example 4 To illustrate the penalized multinomial regression model, here we considered the ordinal data of Example 1, but considering the data as nominal, which under the prediction paradigm may not be so important. To evaluate the performance of this model with Ridge (7.9) and Lasso penalization (7.10), the same 10 random partitions as used in Example 1 were used in the cross-validation strategy. By combining these two penalizations and including different covariates information, the performance of six resulting models were evaluated: penalized Ridge multinomial logistic regression model (PRMLRM) with markers as predictors (PRMLRM-1); PRMLRM with environment and markers as predictors (PRMLRM-2); PRMLRM with environment, markers, and environment×marker interaction as predictors (PRMLRM-3); and the penalized Lasso multinomial logistic regression models, PLMLRM-1, PLMLRM-2, and PLMLRM-3, which have the same predictors as PRMLRM-1, PRMLRM-2, and PRMLRM-3, respectively.

The results are shown in Table 7.4, where again metrics BS and PCCC were computed for each partition. We can observe a similar performance of all models with both metrics. The greater difference in the average BS values across the 10 partitions was found between PRMLRM-3 (worse) model and the PLMLRM-1-PLMRM-2 (best) models, given a 1.34% greater average BS value of the first one compared to the other two. With respect to the other metric, the best average PCCC value (0.29) was obtained with model PRMLRM-2, which gave a 2.36% better performance than the average value of the worse PCCC performance obtained with models PRMLRM-1-2 (0.28).

All these models gave a better performance than all models fitted in Example 1, but only a slightly better performance than the models considered in Example 2, which contain additional information besides the marker information [the only inputs included in the models in Example 1 and PRMLRM-1 and PLMLRG-1], also included environment and environment×marker interaction as predictors. Specifically, even the simpler models in Table 7.4 that use only marker information in the predictor (PRMLRM-1 and PLMLRM-1) gave results comparable to the more complex and competitive models fitted in Example 2. The relative difference in the average BS value between the better models in each of these two sets of compared models (models in Example 2 vs. models in Table 7.4) was 1.25%, while the relative difference in the average PCCC was 3.57%. The R code to reproduce the results of models under Ridge penalization is in Appendix 4. The codes corresponding to the Lasso penalization models are the same as those for Ridge penalization but with the alpha value set to 1 instead of 0, as needs to be done in the basic code of the glmnet to fit the penalized Ridge multinomial logistic regression models:

```
A = cv.glmnet(X, y, family='multinomial', type.measure = "class",
nfolds = 10, alpha = 0)
```

Table 7.4 Brier score (BS) and proportion of cases correctly classified (PCCC) across 10 random partitions, with 80% of the total data set used for training and the rest for testing, under model (7.6), with Ridge and Lasso penalization and different covariable information: the penalized Ridge multinomial logistic regression (PRMLR) model with markers as covariates (PRMLRM-1); the PRMLR model with environment and markers as covariates (PRMLRM-2); the PRMLR model with environment, markers, and environment×marker interaction as covariates (PRMLRM-3); PLMLRM-1, PLRMLRM-2, and PLMLRM-3 denote the penalized Lasso multinomial logistic regression model with the same covariates as in PRMLRM-1, PRMLRM-2, and PRMLRM-3, respectively

Model	PRMLRM-1		PRMLRM-2		PRMLRM-3	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.3755	0.3000	0.3755	0.3000	0.4632	0.3000
2	0.3831	0.3000	0.3831	0.3000	0.3957	0.2667
3	0.3953	0.2667	0.3954	0.2667	0.4052	0.4000
4	0.4093	0.2333	0.4099	0.2333	0.4271	0.0667
5	0.3578	0.3333	0.3578	0.3333	0.3578	0.3333
6	0.3639	0.3333	0.3639	0.3333	0.3639	0.3333
7	0.3885	0.2000	0.3885	0.2000	0.3885	0.2000
8	0.3696	0.3333	0.3700	0.3667	0.3746	0.3333
9	0.3926	0.2333	0.3926	0.2333	0.3926	0.2333
10	0.3955	0.3333	0.3956	0.3333	0.3913	0.2667
Average (SD)	0.3831 (0.01)	0.2866 (0.05)	0.3832 (0.01)	0.29 (0.05)	0.3959 (0.03)	0.2733 (0.09)
Model	PLMLRM-1		PLMLRM-2		PLMLRM-3	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.3644	0.3333	0.3644	0.3333	0.3516	0.4333
2	0.3831	0.3000	0.3831	0.3000	0.3813	0.3000
3	0.3968	0.3000	0.3968	0.3000	0.3996	0.3000
4	0.4147	0.2333	0.4147	0.2333	0.4305	0.1667
5	0.3586	0.3000	0.3586	0.3000	0.3586	0.3000
6	0.3623	0.3000	0.3623	0.3000	0.3667	0.3000
7	0.3885	0.2000	0.3885	0.2000	0.3885	0.2000
8	0.3636	0.3667	0.3636	0.3667	0.3710	0.4000
9	0.3926	0.2333	0.3926	0.2333	0.3926	0.2333
10	0.3913	0.2667	0.3913	0.2667	0.3919	0.2667
Average (SD)	0.3815 (0.01)	0.2833 (0.05)	0.3815 (0.01)	0.2833 (0.05)	0.3832 (0.02)	0.29 (0.08)

where **X** and **y** are the design matrix and vector of response variable for training the multinomial model, and the multinomial model is specified as family='**multinomia**l'; with "**class**" we specified the metric that will be used in the inner cross-validation (CV) strategy to internally select the "optimal" value of the regularization parameter λ (over a grid of 100 values of λ, by default). The misclassification error is equal to $1 - \text{PCCC}$; nfolds is the number of folds that are used internally with the inner CV strategy and when this value is not specified, by default it is set to 10; alpha is a mixing parameter for a more general penalty function

(Elastic Net penalty) that when it is set to 0, the Ridge penalty is obtained, while the Lasso penalty is used when alpha is set to 1; however, when alpha takes values between 0 and 1, the Elastic Net penalization is implemented.

A kind of GBLUP implementation of the simpler models (PRMLRM-1 and PLMLRM-1) considered in Example 4 can be done by considering the genomic relationship matrix derived from the marker information and using as predictor the lower triangular factor of the Cholesky decomposition or its root squared matrix. Something similar can be done to include the Env×Line interaction term. The kinds of "GBLUP" implementation of PRMLRM-1 and PLMLRM-1 are referred to as GMLRM-R1 and GMLRM-L1, respectively. In both cases, the input matrix is $\boldsymbol{X} = \boldsymbol{Z}_L\boldsymbol{L}_g$, where $\boldsymbol{Z}_L$ is the incident matrix of the lines and $\boldsymbol{L}_g$ is the lower triangular part of the Cholesky decomposition of $\boldsymbol{G}$, $\boldsymbol{G} = \boldsymbol{L}_g\boldsymbol{L}_g^{\mathrm{T}}$. For the rest of the models, this kind of "GBLUP" implementation will be referred to as GMLRM-R2 and GMLRM-R3 for Ridge penalty, and as GMLRM-L2 and GMLRM-L3 for Lasso penalty, and in both cases the input matrix $\boldsymbol{X}$ will be $\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g]$ when an environment and genetic effect is present in the predictor (GMLRM-R2, GMLRM-L3), or $\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g, \boldsymbol{Z}_{EL}(\boldsymbol{I}_I \otimes \boldsymbol{L}_g)]$ when the genotype×environment interaction effect is also taken into account (GMLRM-R2, GMLRM-L3), and where I is the number of environments.

The basic code for glmnet implementation of the GMLRM-L3 model is given below, from which the other five kinds of "GBLUP" implementations described above can be performed by only removing the corresponding predictors and changing to 0 the alpha value in the Ridge penalty:

```
Lg = t(chol(G))
#Environment matrix design
XE = model.matrix(~Env,data=dat_F)
#Matrix design of the genetic effect
ZL = model.matrix(~0+GID,data=dat_F)
ZLa = ZL%*%Lg
#Environment-genetic interaction
ZEL = model.matrix(~0+GID:Env,data=dat_F)
ZELa = ZEL%*%kronecker(diag(dim(XE)[2]),Lg)
X = cbind(XE[,-1],ZLa, ZELa)#Input X matrix
A = cv.glmnet(X, y, family='multinomial', type.measure = "class",
              alpha = 1)#alpha=0 for Ridge penalty
```

Example 5 Here we considered the data in Example 4 to illustrate the implementation of the following six kinds of "GBLUP" models: GMLRM-R1, GMLRM-R2, GMLRM-R3, GMLRM-L1, GMLRM-L2, and GMLRM-L3. To evaluate the performance of these models, the same CV strategy was used, where for each of the 10 random partitions, 80% of the full data set was taken to train the models and the rest to evaluate their performance.

Table 7.5 Brier score (BS) and proportion of cases correctly classified (PCCC) across 10 random partitions, with 80% of the data used for training and the rest for testing, for multinomial model (7.6) under the six types of "GBLUP" implementation of the Ridge and Lasso penalization in Example 4 that were obtained by varying the input information: GMLRM-R1 ($\boldsymbol{X} = \boldsymbol{Z}_L\boldsymbol{L}_g$), GMLRM-R2 ($\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g]$), GMLRM-R3 ($\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g, \boldsymbol{Z}_{EL}(\boldsymbol{I}_I \otimes \boldsymbol{L}_g)]$), GMLRM-L1 ($\boldsymbol{X} = \boldsymbol{Z}_L\boldsymbol{L}_g$,), GMLRM-L2 ($\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g]$), and GMLRM-L3 ($\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g, \boldsymbol{Z}_{EL}(\boldsymbol{I}_I \otimes \boldsymbol{L}_g)]$), where the first three use Ridge penalization and the last three Lasso penalty

Model	GMLRM-R1		GMLRM-R2		GMLRM-R3	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.3639	0.3000	0.3625	0.3333	0.3641	0.3333
2	0.3831	0.3000	0.3831	0.3000	0.4187	0.3000
3	0.3705	0.3333	0.3773	0.3000	0.4078	0.2333
4	0.4083	0.2667	0.4083	0.2333	0.4172	0.2667
5	0.3578	0.3333	0.3578	0.3333	0.3578	0.3333
6	0.3639	0.3333	0.3639	0.3333	0.3634	0.3667
7	0.3885	0.2000	0.3885	0.2000	0.3889	0.2000
8	0.3697	0.3667	0.3697	0.3667	0.3875	0.3333
9	0.3926	0.2333	0.3926	0.2333	0.3970	0.2667
10	0.3913	0.2667	0.3913	0.2667	0.3913	0.2667
Average (SD)	0.3789 (0.01)	0.2933 (0.05)	0.3795 (0.01)	0.29 (0.05)	0.3893 (0.02)	0.29 (0.05)
Model	GMLRM-L1		GMLRM-L2		GMLRM-L3	
PT	BS	PCCC	BS	PCCC	BS	PCCC
1	0.3865	0.3667	0.3505	0.4000	0.3598	0.3667
2	0.3831	0.3000	0.3831	0.3000	0.3806	0.3000
3	0.3946	0.2667	0.3946	0.2667	0.3948	0.2667
4	0.4147	0.2333	0.4147	0.2333	0.4147	0.2333
5	0.3564	0.3333	0.3564	0.3333	0.3823	0.2667
6	0.3578	0.3667	0.3578	0.3667	0.3639	0.3333
7	0.4078	0.1333	0.4078	0.1333	0.4291	0.1333
8	0.3670	0.3667	0.3670	0.3667	0.3684	0.3667
9	0.3926	0.2333	0.3926	0.2333	0.3926	0.2333
10	0.3854	0.3333	0.3854	0.3333	0.3913	0.2667
Average (SD)	0.3845 (0.01)	0.2933 (0.07)	0.3809 (0.02)	0.2966 (0.08)	0.3877 (0.02)	0.2766 (0.07)

The results presented in Table 7.5 are comparable to those given in Table 7.4 for Example 4. The relative difference between the models in Table 7.4 with better average BS value (PLMLRM-1) and the model in Table 7.5 with better average BS value (GMLRM-R1) is 0.6990%, in favor of the latter model. Regarding the average PCCC, the "kind" of GBLUP model with better value, GMLRM-L2, was 2.2989% greater than the average PCCC value of the better models in Table 7.4, PMLR-R2 and PMLR-L3. Furthermore, for all models in Table 7.5, the greatest difference between the average BS values was observed in models GMLRM-1 (better) and GMLRM-R3 (worse), a relative difference of 2.748%. The best average PCCC performance was observed with model GMLRM-L2, while the worst one was with model GMLRM-L3, a relative difference of 7.23% between the best and the worst.

The R code to reproduce the results in Table 7.5 for the first three models, see Appendix 5. By only changing the value of alpha to 1, the corresponding R code allows implementing the last three Lasso penalty models.

7.5 Penalized Poisson Regression

In contrast to ordinal data, sometimes the response value is not known to be upper bound, rather it is often a count or frequency data, for example, the number of customers per minute in a specific supermarket, number of infected spikelets per plant, number of defects per unit, number of seeds per plant, etc. The most often used count model to relate a set of explanatory variables with a count response variable is Poisson regression.

Given vector covariates $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}}$, the Poisson log-linear regression modeled the number of events Y_i, as a Poisson random variable with mass density

$$P(Y_i = y|\boldsymbol{x}_i) = \frac{\lambda_i^y \exp(-\lambda_i)}{y!}, y = 0, 1, 2, \ldots, \tag{7.11}$$

where the expected value of Y_i is given by $\lambda_i = \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right)$ and $\boldsymbol{\beta}_0 = (\beta_1, \ldots, \beta_p)^{\mathrm{T}}$ is the vector of beta coefficients effect of the covariates. For example, Y_i may be the number of defects of a product under specific production conditions (temperature, raw material, schedule, operator, etc.) or the number of spikelets per plant which can be predicted from environment and marker information, etc.

For the parameter estimation of this model, the more usual method is maximum likelihood estimation, but this is not suitable in the context of a large number of covariates, so here, as in the logistic and multinomial models, we describe the penalized likelihood method. For a Ridge penalty, the penalized log-likelihood function is given by

$$\begin{aligned}\ell_p(\beta_0, \boldsymbol{\beta}_0; \boldsymbol{y}) &= \sum_{i=1}^{n} \log\left[\, f_{Yi}(y_i; \beta_0, \boldsymbol{\beta}_0)\right] - \lambda \sum_{j=1}^{p} \beta_j^2 \\ &= \sum_{i=1}^{n} y_i\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right) - \sum_{i=1}^{n} \exp\left(\beta_0 + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{\beta}_0\right) - \sum_{i=1}^{n} \log(y_i!) - \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2\end{aligned}$$

This can be optimized directly by using traditional numerical methods, but in a general genomic prediction context where $p \gg n$, other alternatives could be more useful. One is the same procedure described before for multinomial and logistic regressions, in which the log-likelihood is replaced by a second-order approximation at the current beta coefficients and then the resulting weighted least-squares problem is solved by coordinate descent methods. This method for Poisson regression is also

implemented in the glmnet package and the basic code for implementing this is as follows:

```
A = cv.glmnet(X, y, family='poisson', type.measure = "mse", nfolds =
10,
      alpha = 0)
```

where again **X** and **y** are the design matrix and vector of response variable for training the Poisson model which is specified by family='**poisson**'; the metric that will be used internally in the inner cross-validation (CV) strategy to choose the "optimal" regularization parameter λ (over a grid of 100 values of λ, by default) is specified in "type.measure". Additionally, to the mean square error (**mse**), the deviance (**deviance**) and the mean absolute error (**mae**) can also be adopted. The other parameters are the same as described for logistic and multinomial regression models. For implementing the Lasso Poison regression (Poisson regression with Lasso penalty), this same code can be used by only making the argument alpha = 1, instead of 0. Further, in the genomic context, as accomplished before for other kinds of responses (continuous, binary, nominal), with this function more complex models involving marker (genetic), environment, or/and marker × environment interaction effects can also be implemented.

Like the predictors described for the multinomial model, all these predictors are shown in Table 7.6, where the first three (PRPRM-1–3) correspond to Poisson Ridge

Table 7.6 Penalized Poisson regression models with Ridge or Lasso penalties and different covariate information: penalized Ridge Poisson regression (PRPRM) model with markers as covariates (PRPRM-1); PRPRM model with environment and markers as covariates (PRPRM-2); PRPRM model with environment, markers, and environment ×marker interaction as covariates (PRPRM-3); PLPRM-1, PLPRM-2, and PLPRM-3 denote the corresponding penalized Lasso Poisson regression models. GMLRM-R1–R3 are the corresponding GBLUP implementation of the PRPRM models, while GMLRM-L1–L3 are the respective GBLUP of the PLPRM models. $\boldsymbol{X}_E$, $\boldsymbol{X}_M$, and $\boldsymbol{X}_{EM}$ are the design matrices of environment, markers and environment ×marker interaction, $\boldsymbol{Z}_L$ and $\boldsymbol{Z}_{EL}$ are the incident matrices of lines and environment ×line interaction effects, and $\boldsymbol{L}_g$ is the lower triangular part of the Cholesky decomposition of $\boldsymbol{G}$, $\boldsymbol{G} = \boldsymbol{L}_g\boldsymbol{L}_g^T$

Penalization	Model	Predictor effects	Matrix design
Ridge	PRPRM-1	Markers	$\boldsymbol{X} = \boldsymbol{X}_M$
	PRPRM-2	Env + Markers	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{X}_M]$
	PRPRM-3	Env + Markers + Env ×Marker	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{X}_M, \boldsymbol{X}_{EM}]$
Lasso	PLPRM-1	Markers	$\boldsymbol{X} = \boldsymbol{X}_M$
	PLPRM-2	Env + Markers	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{X}_M]$
	PLPRM-3	Env + Markers + Env ×Marker	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{X}_M, \boldsymbol{X}_{EM}]$
	GPRM-R1	Genetic	$\boldsymbol{X} = \boldsymbol{Z}_L\boldsymbol{L}_g$
Ridge	GPRM-R2	Env + Genetic	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g]$
	GPRM-R3	Env + Genetic + Env ×Genetic	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g, \boldsymbol{Z}_{EL}(\boldsymbol{I}_I \otimes \boldsymbol{L}_g)]$
	GPRM-L1	Genetic	$\boldsymbol{X} = \boldsymbol{Z}_L\boldsymbol{L}_g$
Lasso	GPRM-L2	Env + Genetic	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g]$
	GPRM-L3	Env + Genetic + Env× Genetic	$\boldsymbol{X} = [\boldsymbol{X}_E, \boldsymbol{Z}_L\boldsymbol{L}_g, \boldsymbol{Z}_{EL}(\boldsymbol{I}_I \otimes \boldsymbol{L}_g)]$

regression with marker, environment plus marker, and environment plus markers plus environment×marker interaction effects, respectively. The second group of three models (PLPRM-1–3) are, respectively, the same as before but with the Lasso penalization. The third group—three models (GRPRM-R1–R3)—are the corresponding GBLUP implementation of the first three predictors, while the last group—three models (GLPRM-L1–L3)—corresponds to the GBLUP implementation of the second group—three Lasso penalized models.

Example 6 To illustrate how to fit the Poisson model with the predictors given in Table 7.6 and compare these in terms of the mean squared error of prediction, we considered a small data set that is part of the data set used in Montesinos-López et al. (2016). It consists of 50 lines in three environments and a total of 1635 markers. The performance evaluation was done using 10 random partitions where 80% of the data was used to train the model and the rest to test the model. The glmnet R package was used to train all these regression models.

The results are shown in Table 7.7, where the first six models are the first six penalized Poisson regression models given in Table 7.6. With respect to the Ridge penalized models (PRPRM-1–3), we can observe a slightly better performance of the more complex models that take into account environment, marker, and environment ×marker interaction effects (PRPRM-3). As for the penalized Lasso Poisson regression models (PLPRM-1–3), but with a more notable difference, the more complex models also resulted in better performance, and this was also better than the penalized Ridge model (PRPRM-3).

Furthermore, in its GBLUP implementation of the Poisson regression models with Ridge penalty, better MSE performance was obtained with the model that includes environment and genetic effects (GPRM-R2) and this was the best among all Ridge penalized models (PRPRM-1–3 and GPRM-R1–R3); the average MSE of the best PRPRM (PRPRM-3) is 7.5668 greater than the average MSE of the best GPRM with Ridge penalty (GPRM-R2). Under the GBLUP implementations of the Poisson regression models with Lasso penalty (GPRM-L1–L3), the best performance prediction was also obtained when using the environment and genetic effects (GPRM-R2), but its average MSE was slightly different (by only 2.71%) than that obtained with the average MSE of the best Lasso penalized Poisson regression model (PLPRM-3). Additionally, this GBLUP implementation (GPRM-L2) also showed the best average MSE performance among all 12 implemented models, and a notable difference with the second best (GPRM-R2) that gave a relative MSE that was 12.01% greater than that for GPRM-L2 model.

The worse average MSE performance of model **GPRM-L3** is because of the high value of the MSE obtained in partition 2. However, this high variability of the MSEP observed across partitions can be unexpected for other larger data sets.

The R code to reproduce these results is shown in Appendix 6.

Table 7.7 Mean square error of prediction (MSE) of six penalized Poisson regression models with Ridge or Lasso penalties and different covariate information (first six models) and MSE of the GBLUP implementation of the corresponding models (second six models), across 10 random partitions, with 80% of the total data set used for training and the rest for testing. See Table 7.6 for details of each model

Model	PRPRM-1	PRPRM-2	PRPRM-3	PLPRM-1	PLPRM-2	PLPRM-3
PT	MSE	MSE	MSE	MSE	MSE	MSE
1	3.7346	3.7380	3.5678	3.7573	3.0309	3.0019
2	2.0492	2.1012	2.1012	2.1194	2.0408	2.0511
3	2.6610	2.6757	2.7178	2.6642	2.1885	2.1930
4	4.6873	4.7079	4.4640	4.5714	4.3558	3.8514
5	3.4277	3.4252	3.3573	3.3317	2.5760	2.6570
6	2.6719	2.6589	2.6592	2.9291	2.1381	2.1104
7	2.7093	2.6601	2.6601	2.6601	3.2845	2.4267
8	3.8690	3.8791	3.7156	3.7156	3.1182	3.1605
9	1.9520	1.9699	1.8081	1.8081	1.5642	1.7106
10	4.6182	4.6086	4.5979	4.6416	3.6876	3.8178
Average (SD)	3.238 (0.98)	3.2424 (0.97)	3.1648 (0.93)	3.2198 (0.96)	2.7984 (0.85)	2.698 (0.74)
Model	GPRM-R1	GPRM-R2	GPRM-R3	GPRM-L1	GPRM-L2	GPRM-L3
PT	MSE	MSE	MSE	MSE	MSE	MSE
1	3.7110	3.4279	3.8221	3.8243	2.8399	2.9275
2	2.1012	1.7678	2.1050	2.1012	1.8690	69.3985
3	2.6606	2.2621	2.7178	2.7178	2.0578	1.9326
4	4.7780	4.8127	4.2797	4.2820	3.9353	3.3327
5	3.5584	3.0863	3.5038	3.3910	2.6122	2.6139
6	2.6117	2.0641	2.8482	2.7042	2.0844	1.9349
7	2.7123	2.5304	2.9432	2.6601	2.7140	2.3736
8	3.7156	3.5213	3.7107	3.6467	2.9335	3.0768
9	1.8100	1.6769	1.8005	1.8081	1.4778	1.6548
10	4.5861	4.2733	4.5821	4.8303	3.7439	4.1025
Average (SD)	3.2244 (1.0)	2.9422 (1.06)	3.2312 (0.9)	3.1965 (0.96)	2.6267 (0.79)	9.3347 (21.11)

7.6 Final Comments

In this chapter, we give the fundamentals of some popular Bayesian and classical regression models for categorical and count data. We also provide many practical examples for implementing these models for genomic prediction. The examples used take into account in the predictor not only the effect of markers or genotypes but also illustrate how to take into account the effects of environment, genotype × environment interaction, and marker × environment interaction. Also, for each type of response variables, we calculated the prediction performance using appropriate

metrics, which is very important since the metrics for evaluating the prediction performance are dependent upon the type of response variable. It is important to point out that the components to include in the predictor are not restricted to those illustrated here, since the user can include other components as main effects or interaction effects in similar fashion as illustrated in the examples provided.

Appendix 1

```
rm(list=ls(all=TRUE))
library(BGLR)
load('dat_ls.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
#Marker information
dat_M = dat_ls$dat_M
#Trait response values
y = dat_F$y
summary(dat_F)
#Sorting data phenotypic data set first by Environment, and then by GID
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
#10 random partitions
K = 10; n = length(y)
set.seed(1)
PT = replicate(K,sample(n,0.20*n))

#Brier score function
BS_f<-function(y,p)
{
 n = length(y)
 CM = matrix(0,nr=n,nc=dim(p)[2])
 CM[cbind(1:n,y)]<-1
 mean(rowSums((p-CM)^2))/2
}
#Matrix design of markers for all observations
Pos = match(dat_F$GID,row.names(dat_M))
X = dat_M[Pos,]
X = scale(X)
#Ordinal BRR model
ETA_BRR = list(list(X=X,model='BRR'))
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_BRR,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
```

```
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
}
Tab
#Ordinal Bayesian GBLUP model
#Genomic relationship matrix
X_M = scale(dat_M)
G = (X_M%*%t(X_M))/dim(X_M)[2]
dat_F$GID = factor(dat_F$GID,levels=colnames(G))
Z_L = model.matrix(~0+GID,data=dat_F)
Ga = Z_L%*%G%*%t(Z_L)
ETA_GBLUP = list(list(K=Ga,model='RKHS'))
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_GBLUP,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
}
Tab

#Ordinal BayesA model
ETA_BA = list(list(X=X,model='BayesA'))
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_BA,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
}
Tab
#Ordinal BayesB model
ETA_BB = list(list(X=X,model='BayesB'))
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_BB,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
```

```
}
Tab
#Ordinal BayesC model
ETA_BC = list(list(X=X,model='BayesC'))
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_BC,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
}
Tab

#Ordinal BL model
ETA_BL = list(list(X=X,model='BL'))
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_BL,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
}
Tab
```

Appendix 2

```
rm(list=ls(all=TRUE))
library(BGLR)
load('dat_ls.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
#Marker information
dat_M = dat_ls$dat_M
y = dat_F$y
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
#PT
#10 random partitions
K = 10
n = length(y)
set.seed(1)
```

```
PT = replicate(K,sample(n,0.20*n))
#Brier score
BS_f<-function(y,p)
{
 n = length(y)
 CM = matrix(0,nr=n,nc=dim(p)[2])
 CM[cbind(1:n,y)]<-1
 mean(rowSums((p-CM)^2))/2
}
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
X = dat_M[Pos,]
X = scale(X)

#Environment matrix design
XE = model.matrix(~Env,data=dat_F)[,-1]
#Environment-marker interaction
Envs = unique(dat_F$Env)
Markers = colnames(X)
XEM = model.matrix(~0+X:Env,data=dat_F)
Pos = !(colnames(XEM)%in%paste0('X',Markers,':Env',Envs[length
(Envs)]))
XEM = XEM[,Pos]

#Models to evaluate
Models = c('BRR','BayesA','BayesB','BayesC','BL')
for(m in 1:5)
{
 ETA = list(list(X=XE,model='FIXED'),list(X=X,model=Models[m]),
       list(X=XEM,model=Models[m]))
 Tab = data.frame(PT = 1:K,BS=NA)
 set.seed(1)
 for(k in 1:K)
 {
  Pos_tst = PT[,k]
  y_NA = y
  y_NA[Pos_tst] = NA
  A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA,
       nIter = 1e4,burnIn = 1e3,verbose = FALSE)
  Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
  Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
  if(dim(A$probs)[2]<5) stop
 }
#Saving the output (Tab) for each model
 write.csv(Tab,file=paste0('Tab_M3',Models[m],'.csv'),row.names =
FALSE)
}

#GBLUP regression model (4)
#Genomic relationship matrix
G = (dat_M%*%t(dat_M))/dim(dat_M)[2]
dat_F$GID = factor(dat_F$GID,levels=colnames(G))
Z_L = model.matrix(~0+GID,data=dat_F)
```

```
Ga = Z_L%*%G%*%t(Z_L)
#"Genomic" relationship matrix for interaction
Ga2 = kronecker(diag(3),G)
ETA_GBLUP = list(list(K=Ga,model='RKHS'),
         list(X=XE,model='FIXED'),
         list(K=Ga2,model='RKHS'))
Tab = data.frame(PT = 1:K,BS=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_GBLUP,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst],A$probs[Pos_tst,])
 Tab$PCC[k] = mean(y[Pos_tst]==apply(A$probs,1,which.max)[Pos_tst])
}
Tab
```

Appendix 3

```
rm(list=ls(all=TRUE))
library(BGLR)
load('Data_Toy_EYT.RData',verbose=TRUE)
#Phenotypic data
dat_F = Pheno_Toy_EYT
head(dat_F)
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
y = dat_F$Height
#Genomic relationship matrix
G = G_Toy_EYT
#PT
#10 random partitions
K = 10
n = length(y)
set.seed(1)
PT = replicate(K,sample(n,0.20*n))
#Brier score
#The min value of y must be 1
BS_f<-function(y,p)
{
 n = length(y)
 CM = matrix(0,nr=n,nc=dim(p)[2])
 CM[cbind(1:n,y)]<-1
 mean(rowSums((p-CM)^2))/2
}
#M3
dat_F$GID = factor(dat_F$GID,levels=colnames(G))
Z_L = model.matrix(~0+GID,data=dat_F)
```

```
Ga = Z_L%*%G%*%t(Z_L)
ETA_GBLUP = list(list(K=Ga,model='RKHS'))
#GBLUP
Tab = data.frame(PT = 1:K,BS=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_GBLUP,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst]+1,A$probs[Pos_tst,])
 lbs = as.numeric(colnames(A$probs))
 yp = apply(A$probs,1,function(x)lbs[which.max(x)])
 Tab$PCC[k] = mean(y[Pos_tst]==yp[Pos_tst])
}
Tab

#M4
#Environment matrix design
XE = model.matrix(~Env,data=dat_F)[,-1]
#"Genomic" relationship matrix for interaction
Ga2 = kronecker(diag(4),G)
ETA_GBLUP = list(list(X=XE,model='FIXED'),
         list(K=Ga,model='RKHS'),
         list(K=Ga2,model='RKHS'))
#GBLUP
Tab = data.frame(PT = 1:K,BS=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_GBLUP,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst]+1,A$probs[Pos_tst,])
 lbs = as.numeric(colnames(A$probs))
 yp = apply(A$probs,1,function(x)lbs[which.max(x)])
 Tab$PCC[k] = mean(y[Pos_tst]==yp[Pos_tst])
}
Tab

#M5
#Environment matrix design
XE = model.matrix(~Env,data=dat_F)[,-1]
#mean(G!=Ga2)
ETA_GBLUP = list(list(K=Ga,model='RKHS'),
         list(X=XE,model='FIXED'))
Tab = data.frame(PT = 1:K,BS=NA)
set.seed(1)
for(k in 1:K)
```

```
{
 Pos_tst = PT[,k]
 y_NA = y
 y_NA[Pos_tst] = NA
 A = BGLR(y=y_NA,response_type='ordinal',ETA=ETA_GBLUP,
      nIter = 1e4,burnIn = 1e3,verbose = FALSE)
 Tab$BS[k]  = BS_f(y[Pos_tst]+1,A$probs[Pos_tst,])
 lbs = as.numeric(colnames(A$probs))
 yp = apply(A$probs,1,function(x)lbs[which.max(x)])
 Tab$PCC[k] = mean(y[Pos_tst]==yp[Pos_tst])
}
Tab
```

Appendix 4 (Example 4)

```
rm(list=ls(all=TRUE))
library(glmnet)
load('dat_ls.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
#Marker information
dat_M = dat_ls$dat_M
#Response variable
y = dat_F$y
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
head(dat_F)
#PT
#10 random partitions
K = 10
n = length(y)
set.seed(1)
PT = replicate(K,sample(n,0.20*n))

#Brier score function
BS_f<-function(y,p)
{
 n = length(y)
 CM = matrix(0,nr=n,nc=dim(p)[2])
 CM[cbind(1:n,y)]<-1
 mean(rowSums((p-CM)^2))/2
}
#PRMLRM-1
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
X = dat_M[Pos,]
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
```

```
 y_tr = as.character(y[-Pos_tst])
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='multinomial',
        type.measure = "class",# Misclassification error
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0) # alpha=1 for lasso penalization
 p_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)[,,1]
 y_tst = y[Pos_tst]
 Tab$BS[k]  = BS_f(y_tst,p_tst)
 yp_tst = predict(A,newx = X[Pos_tst,],type='class',s=A$lambda.min)
 yp_tst = as.numeric(yp_tst)
 Tab$PCC[k] = mean(y_tst==yp_tst)
}
Tab
#PRMLRM-2
# Marker + Env effect
#Environment_marker interaction
XE = model.matrix(~Env,data=dat_F)[,-1]
#Environment matrix design
XEM = model.matrix(~0+X:Env,data=dat_F)
dim(XEM)
head(XEM)[,1:5]
#XEMa = kronecker(diag(3),unique(X))
#sum((XEM-XEMa)^2)
#Matrix desing for PMR
X = cbind(XE,X)
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = as.character(y[-Pos_tst])
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='multinomial',
        type.measure = "class",
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0) # alpha=1 for lasso penalization
 p_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)[,,1]
 y_tst = y[Pos_tst]
 Tab$BS[k]  = BS_f(y_tst,p_tst)
 yp_tst = predict(A,newx = X[Pos_tst,],type='class',s=A$lambda.min)
 yp_tst = as.numeric(yp_tst)
 Tab$PCC[k] = mean(y_tst==yp_tst)
}
Tab
#PRMLRM-3
#Marker + Env + Env:Marker effect
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
X = dat_M[Pos,]
#Environment_marker interaction
```

```
XEM = model.matrix(~0+X:Env,data=dat_F)
X = cbind(XE,X,XEM)
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = as.character(y[-Pos_tst])
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='multinomial',
        type.measure = "class",
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0) # alpha=1 for lasso penalization
 #plot(A)
 p_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)[,,1]
 y_tst = y[Pos_tst]
 Tab$BS[k]  = BS_f(y_tst,p_tst)
 yp_tst = predict(A,newx = X[Pos_tst,],type='class',s=A$lambda.min)
 yp_tst = as.numeric(yp_tst)
 Tab$PCC[k] = mean(y_tst==yp_tst)
}
Tab
```

Appendix 5

```
rm(list=ls(all=TRUE))
library(glmnet)
load('dat_ls.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
#Marker information
dat_M = dat_ls$dat_M
y = dat_F$y
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
#PT
#10 random partitions
K = 10
n = length(y)
set.seed(1)
PT = replicate(K,sample(n,0.20*n))
#Brier score function
BS_f<-function(y,p)
{
 n = length(y)
 CM = matrix(0,nr=n,nc=dim(p)[2])
 CM[cbind(1:n,y)]<-1
 mean(rowSums((p-CM)^2))/2
}
#GMLRM-R1: Genomic effects
```

```
#Matrix design of markers
X  = dat_M
#GIDs = sort(row.names(X))
#X  = X[match(GIDs,row.names(X)),]
G  = tcrossprod(X)/dim(X)[2]
dat_F$GID = factor(dat_F$GID,levels=row.names(G))
ZL = model.matrix(~0+GID,data=dat_F)
Lg = t(chol(G))#Lower triangular part of the Cholesky descomposition of
G
#Design matrix to use with glmnet
ZLa  = ZL%*%Lg
X =  ZLa
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = as.character(y[-Pos_tst])
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='multinomial',
        type.measure = "class",# Misclassification error
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0)#take alpha=1 for lasso penalty, GMLRM-L1
 #Prediction of testing set
 p_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)[,,1]
 y_tst = y[Pos_tst]
 Tab$BS[k]  = BS_f(y_tst,p_tst)
 yp_tst = predict(A,newx = X[Pos_tst,],type='class',s=A$lambda.min)
 yp_tst =  as.numeric(yp_tst)
 Tab$PCC[k] = mean(y_tst==yp_tst)
}
Tab
#GMLRM-R2: Genomic + Env effect
#Matrix design of markers
ZLa  = ZL%*%Lg
#Environment_marker interaction
XE =  model.matrix(~Env,data=dat_F)
#Matrix design to use in glmnet
X = cbind(XE[,-1],ZLa)
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = as.character(y[-Pos_tst])
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='multinomial',
        type.measure = "class",
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0)# take alpha=1 for lasso penalty, GMLRM-L2
 p_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)[,,1]
```

```
 y_tst = y[Pos_tst]
 Tab$BS[k]  = BS_f(y_tst,p_tst)
 #Prediction of testing set
 yp_tst = predict(A,newx = X[Pos_tst,],type='class',s=A$lambda.min)
 yp_tst = as.numeric(yp_tst)
 Tab$PCC[k] = mean(y_tst==yp_tst)
}
Tab

#GMLRM-R3: Genomic + Env effect + Env:Marker effect
#Matrix design of markers
ZLa = ZL%*%Lg
#Environment-genetic interaction
ZEL = model.matrix(~0+GID:Env,data=dat_F)
ZELa = ZEL%*%kronecker(diag(dim(XE)[2]),Lg)
#Input matrix to use in glmnet
X = cbind(XE,ZLa,ZELa)
Tab = data.frame(PT = 1:K,BS=NA,PCC=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = as.character(y[-Pos_tst])
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='multinomial',
        type.measure = "class",
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0)# alpha=1 for lasso penalty, GMLRM-L3
 p_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)[,,1]
 y_tst = y[Pos_tst]
 Tab$BS[k]  = BS_f(y_tst,p_tst)
  #Prediction of testing set
 yp_tst = predict(A,newx = X[Pos_tst,],type='class',s=A$lambda.min)
 yp_tst = as.numeric(yp_tst)
 Tab$PCC[k] = mean(y_tst==yp_tst)
}
Tab
```

Appendix 6

```
rm(list=ls(all=TRUE))
library(glmnet)
load('dat_ls.RData',verbose=TRUE)
#Phenotypic data
dat_F = dat_ls$dat_F
#Marker information
dat_M = dat_ls$dat_M
y = dat_F$y
dat_F$Env= as.character(dat_F$Loc)
```

```
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
head(dat_F)
#PT
#10 random partitions
K = 10
n = length(y)
set.seed(1)
PT = replicate(K,sample(n,0.20*n))
#Penalized Ridge (lasso) Poisson regression models
#PRPRM-1-3 (PLPRM-1-3)
#Matrix design of markers
Pos = match(dat_F$GID,row.names(dat_M))
X = dat_M[Pos,]
#Environment matrix design
XE = model.matrix(~Env,data=dat_F)[,-1]
#Environment-marker interaction
Envs = unique(dat_F$Env)
Markers = colnames(X)
XEM = model.matrix(~0+X:Env,data=dat_F)
Pos = !(colnames(XEM)%in%paste0('X',Markers,':Env',Envs[length
(Envs)]))
XEM = XEM[,Pos]
X = X# Design matrix for PRPRM-1 and PLPRM-1
#X = cbind(XE,X) # Uncomment this line to implement PRPRM-2 or PLPRM-2
#X = cbind(XE,X,XEM) # Uncomment this line to implement PRPRM-3 or PLPRM-
3
Tab = data.frame(PT = 1:K,MSEP=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = y[-Pos_tst]
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='poisson',
        type.measure = "mse",
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0)# Take alpha=1 for lasso penalty
 yp_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)
 y_tst = y[Pos_tst]
 Tab$MSEP[k]  = mean((y_tst-yp_tst)^2)

}
Tab

#GBLUP implementation of penalized Ridge (lasso) Poisson regression
models
#GPRM-R1-R3 (GPRM-L1-L3)
#Matrix design of Environments
XE = model.matrix(~Env,data=dat_F)
#Matrix design of markers
X  = dat_M
G  = tcrossprod(X)/dim(X)[2] # Genomic relationship matrix
```

```
dat_F$GID = factor(dat_F$GID,levels=row.names(G))
dat_F = dat_F[order(dat_F$Env,dat_F$GID),]
ZL = model.matrix(~0+GID,data=dat_F)
mean(substring(colnames(ZL),4)!=row.names(G))
Lg = t(chol(G))
#Matrix design of genetic effects
ZLa = ZL%*%Lg
#Environment-genetic interaction matrix design
ZEL = model.matrix(~0+GID:Env,data=dat_F)
ZELa = ZEL%*%kronecker(diag(dim(XE)[2]),Lg)
X = ZLa # Matrix design for GPRM-R1 and GPRM-L1
#X = cbind(XE[,-1],ZLa) # Uncomment this line to implement GPRM-R2 or
GPRM-L2
#X = cbind(XE[,-1],ZLa,ZELa) ) # Uncomment this line to implement GPRM-
R3 or GPRM-L3
Tab = data.frame(PT = 1:K,MSEP=NA)
set.seed(1)
for(k in 1:K)
{
 Pos_tst = PT[,k]
 y_tr = y[-Pos_tst]
 X_tr = X[-Pos_tst,]
 A = cv.glmnet(X_tr, y_tr, family='poisson',
        type.measure = "mse",
        nfolds = 10, parallel = FALSE,thresh=1e-10,
        alpha = 0) # Take alpha=1 for lasso penalty
 yp_tst = predict(A,newx = X[Pos_tst,],type='response',s=A$lambda.
min)
 y_tst = y[Pos_tst]
 Tab$MSEP[k]  = mean((y_tst-yp_tst)^2)
}
Tab
```

References

Agresti A (2012) Categorical data analysis, 3rd edn. Wiley, Hoboken, NJ

Albert JH, Chib S (1993) Bayesian analysis of binary and polychotomous response data. J Am Stat Assoc 88(422):669–679

Friedman J, Hastie T, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. J Stat Softw 33(1):1–22. http://www.jstatsoft.org/v33/i01/

Gianola D (1980) A method of sire evaluation for dichotomies. J Anim Sci 51(6):1266–1271

Gianola D (1982) Theory and analysis of threshold characters. J Anim Sci 54(5):1079–1096

Gianola D, Foulley JL (1983) Sire evaluation for ordered categorical data with a threshold model. Genet Select Evol 15(2):201–224

McCullagh P (1980) Regression models for ordinal data. J R Stat Soc B Methodol 42(2):109–142

Montesinos-López OA, Montesinos-López A, Pérez-Rodríguez P, de los Campos G, Eskridge K, Crossa J (2015a) Threshold models for genome-enabled prediction of ordinal categorical traits in plant breeding. G3 5(2):291–300

Montesinos-López OA, Montesinos-López A, Crossa J, Burgueño J, Eskridge K (2015b) Genomic-enabled prediction of ordinal data with Bayesian logistic ordinal regression. G3 5(10):2113–2126

Montesinos-López A, Montesinos-López OA, Crossa J, Burgueño J, Eskridge KM, Falconi-Castillo E et al (2016) Genomic Bayesian prediction model for count data with genotype × environment interaction. G3 6(5):1165–1177

Pérez P, de los Campos, G. (2014) BGLR: a statistical package for whole genome regression and prediction. Genetics 198(2):483–495

Polson NG, Scott JG, Windle J (2013) Bayesian inference for logistic models using Pólya–Gamma latent variables. J Am Stat Assoc 108(504):1339–1349

Sorensen DA, Andersen S, Gianola D, Korsgaard I (1995) Bayesian inference in threshold models using Gibbs sampling. Genet Sel Evol 27(3):1–21

Zhu J, Hastie T (2004) Classification of gene microarrays by penalized logistic regression. Biostatistics 5(3):427–443

Chapter 8
Reproducing Kernel Hilbert Spaces Regression and Classification Methods

8.1 The Reproducing Kernel Hilbert Spaces (RKHS)

One of the main goals of genetic research is accurate phenotype prediction. This goal has largely been achieved for Mendelian diseases with a small number of risk variants (Schrodi et al. 2014). However, many traits (like grain yield) have a complex genetic architecture that is not well understood (Golan and Rosset 2014). Phenotype prediction for such traits remains a major challenge. A key challenge in complex phenotype prediction is accurate modeling of genetic interactions, commonly known as epistatic effects (Cordell 2002). In recent years, there has been mounting evidence that epistatic interactions are widespread throughout biology (Moore and Williams 2009; Lehner 2011; Hemani et al. 2014; Buil et al. 2015). It is well accepted that epistatic interactions are biologically plausible, on the one hand (Zuk et al. 2012), and are difficult to detect, on the other hand (Cordell 2009), suggesting that they may be highly influential in our limited success in modeling complex heritable traits.

Reproducing Kernel Hilbert Spaces (RKHS) regression was one of the earliest statistical machine learning methods suggested for use in plant and animal breeding (Gianola et al. 2006; Gianola and van Kaam 2008) for the prediction of complex traits. An RKHS is a Hilbert space of functions in which all the evaluation functionals are bounded linear functionals. The fundamental idea of RKHS methods is to project the given original input data contained in a finite-dimensional vector space onto an infinite-dimensional Hilbert space. The kernel method consists of transforming the data using a kernel function and then applying conventional statistical machine learning techniques to the transformed data, hoping for better results. Methods based on implicit transformations (RKHS methods) have become very popular for analyzing nonlinear patterns in data sets from various fields of study. Furthermore, the introduction of kernel functions has become an efficient alternative to obtain measures of similarity between objects that do not have a natural vector representation. Although the best known application of kernel methods is

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_8

Support Vector Machines (SVM), which is studied in the next chapter, lately it has been shown that any learning algorithm based on distances between objects can be formulated in terms of kernel functions, applying the so-called "kernel trick." However, RKHS methods are not limited to regression; they are also really powerful for classification and data compression problems and theoretically sound for dealing with nonlinear phenomena in general. For these reasons, they have found a wide range of practical applications ranging from bioinformatics to text categorization, from image analysis to web retrieval, from 3D reconstruction to handwriting recognition, and from geostatistics to chemoinformatics. The increase in popularity of kernel-based methods is also due in part to the fact that they provide a rich way to capture nonlinear patterns in data that cannot be captured with conventional linear statistical learning methods. In genomic selection, the application of RKHS methods continues to increase, for example, Long et al. (2010) found better performance of RKHS methods over linear models in body weight of broiler chickens. Crossa et al. (2010) compared RKHS versus Bayesian Lasso and found that RKHS was better than Bayesian Lasso in the wheat data set, but a similar performance of both methods was observed in the maize data set. Cuevas et al. (2016, 2017, 2018) found superior performance of RKHS methods over linear models using Gaussian kernels on data of maize and wheat. Cuevas et al. (2019) also found that when using pedigree, markers, and near-infrared spectroscopy (NIR) data (which is an inexpensive and nondestructive high-throughput phenotyping technology for predicting unobserved line performance in plant breeding trials), kernel methods (Gaussian kernel and arc-cosine kernel) outperformed linear models in terms of prediction performance. However, other authors found minimal differences between RKHS methods and linear models, for example, Tusell et al. (2013) in litter size in swine, Long et al. (2010) and Morota et al. (2013) in progeny tests of dairy sires, and Morota et al. (2014) in phenotypes of dairy cows. These publications have empirically shown equal or better prediction ability of RKHS methods over linear models. For this reason, the applications of kernel methods in GS are expected to continue increasing since they can be implemented in current software of genomic prediction and because they are (a) very flexible, (b) easy to interpret, (c) theoretically appealing for accommodating cryptic forms of gene action (Gianola et al. 2006; Gianola and van Kaam 2008), (d) these methods can be used with almost any type of information (e.g., covariates, strings, images, and graphs) (de los Campos et al. 2010), (e) computation is performed in an n-dimensional space even when the original input information has more columns (p) than observations (n) thus avoiding the $p \gg n$ problem (de los Campos et al. 2010), (f) they provide a new viewpoint whose full potential is still far from our understanding, and (g) they are very attractive due to their computational efficiency, robustness, and stability.

The goal of this chapter is to give the user (student or scientist) a friendly introduction to regression and classification methods based on kernels. We also cover the essentials of kernels methods, and with examples, we show the user how to handcraft an algorithm of a kernel for applications in the context of genomic selection.

8.2 Generalized Kernel Model

Like any regression problem, a generalized kernel model assumes that we have pairs $(y_i, \boldsymbol{x}_i)$ for $i = 1, \ldots, n$, where y_i and $\boldsymbol{x}_i$ are the response variable and the vector of independent variables (pedigree of marker data) measured in individual i, and the relationship between y_i and $\boldsymbol{x}_i$ is given by

$$\begin{aligned} \text{Distribution} &: y_i \sim p(y_i|\mu_i) \\ \text{Linear predictor} &: \eta_i = f(\boldsymbol{x}_i) = \eta_0 + \boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta} \\ \text{Link function} &: \eta_i = g(\mu_i) \end{aligned}$$

where $g(.)$ is a known link function, $\mu_i = h(\eta_i)$, $h(.)$ denotes the inverse link function, $f(\boldsymbol{x}_i) = \eta_0 + \boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta}, \eta_0$ is an intercept term, $\boldsymbol{k}_i = [K(\boldsymbol{x}_i, \boldsymbol{x}_1), \ldots, K(\boldsymbol{x}_i, \boldsymbol{x}_n)]^{\mathrm{T}}$, $K(.,.)$ is the kernel function, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)^{\mathrm{T}}$ is an $n \times 1$ vector of coefficients. This generalized kernel model provides a unifying framework for kernel-based analyses for dealing with continuous, binary, categorical, and count data, since with different $p(y_i|\mu_i)$ and $g(.)$, we have different models. It is very interesting to point out that under the kernel framework, the problem is reduced to finding n regression coefficients instead of p, as in conventional regression models, thus avoiding the problem of having to solve a regression problem with $p \gg n$. Also, kernel methods are very useful when genotypes and phenotypes are connected in ways that are not well addressed by the linear additive models that are standard in quantitative genetics.

8.2.1 Parameter Estimation Under the Frequentist Paradigm

Inferring f requires defining a collection (or space) of functions from which an element, $\widehat{f}$, will be chosen via a criterion. Specifically, in RKHS, estimates are obtained by solving the following optimization problem:

$$\underbrace{\min}_{f \in H} \left\{ \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)) + \lambda \| f \|_H^2 \right\}, \tag{8.1}$$

which mean that the optimization problem is performed within the space of functions H, a RKHS, $f \in H$ and $\|f\|_H$ denotes the norm of f in Hilbert space H; $L(y_i, f(\boldsymbol{x}_i))$ is some measure of goodness of fit, that is, a loss function viewed as the negative conditional log-likelihood, which should be chosen in agreement with the type of response variable. For example, for continuous outcomes, this should be constructed in terms of Gaussian distributions, when the response variable is binary in terms of Bernoulli distributions, when the response is count in terms of Poisson or negative

binomial distribution, and when it is categorical in terms of multinomial distributions; λ is a smoothing or regularization parameter that should be positive and should control the trade-off between model goodness of fit and complexity; and $\| f\|_H^2$ is the square of the norm of $f(\boldsymbol{x}_i)$ on H, a measure of model complexity (de los Campos et al. 2010). Hilbert spaces are complete linear spaces endowed with a norm that is the square root of the inner product in the space. The Hilbert spaces that are relevant for our discussion are RKHS of real-valued functions, here denoted as H. Those interested in more technical details of RKHS of real functions should read Wahba (1990). By the representer theorem (Wahba 1990), which tells us that the solutions to some regularization functionals in high or infinite-dimensional spaces fall in a finite-dimensional space, the solution for (8.1) admits a linear representation

$$f(\boldsymbol{x}_i) = \eta_0 + \sum_{j=1}^{n} \beta_j K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \eta_0 + \boldsymbol{k}_i^{\mathrm{T}} \boldsymbol{\beta}, \tag{8.2}$$

where η_0 is an intercept term, $K(\cdot, \cdot)$ is the kernel function, $k_{\mathrm{i}} = [K(x_i, x_1), \ldots, K(x_i, x_{\mathrm{n}})]^{\mathrm{T}}$ as defined before, and β_j are beta coefficients. Notice that $\| f\|_H^2 = \sum_{l,j=1}^{n} \beta_l \beta_j K\left(\boldsymbol{x}_l, \boldsymbol{x}_j\right)$, and by substituting (8.2) into (8.1), we obtain the minimization problem under a frequentist framework with respect to η_0 and $\boldsymbol{\beta}$, as did Gianola et al. (2006) and Zhang et al. (2011):

$$\underbrace{\min}_{\eta_0, \boldsymbol{\beta}} \left\{ \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \eta_0 + \boldsymbol{k}_i^{\mathrm{T}} \boldsymbol{\beta}\right) + \frac{\lambda}{2} \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\beta} \right\}, \tag{8.3}$$

where $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n]$ is the $n \times n$ kernel matrix with $\boldsymbol{k}_i$ as defined above. Since $\boldsymbol{K}$ needs to be symmetric and positive semi-definite, the term $\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\beta}$ is an empirical RKHS norm with regard to the training data, λ is a smoothing or regularization parameter that should be positive and should control the trade-off between model goodness of fit and complexity, and the factor $\frac{1}{2}$ is introduced for convenience. The second term of (8.3) acts as a penalization term that is added to the minus log-likelihood. The goal is to find η_0 and $\boldsymbol{\beta}$, which is equivalent to finding $f(\boldsymbol{x}_i) = \eta_0 + \boldsymbol{k}_i^{\mathrm{T}} \boldsymbol{\beta}$ that minimizes (8.3). $f(\boldsymbol{x}_i)$ is based on a basis expansion of kernel functions and this relationship $f(\boldsymbol{x}) = \eta_0 + \boldsymbol{k}_i^{\mathrm{T}} \boldsymbol{\beta}$ is due to the representer theorem (Wahba 1990). Therefore, model specification under the generalized RKHS methods depends on the choice of loss function $L(..)$, the Hilbert space H to build $\boldsymbol{K}$, and the smoothing parameter λ. The smoothing parameter λ can be chosen by cross-validation or generalized cross-validation under the frequentist framework or by specifying a prior distribution for the $\boldsymbol{\beta}$ coefficients under the Bayesian framework (Gianola and van Kaam 2008). It is important to point out that when the response variable is coded as $y_i \in \{-1, 1\}$ and the hinge function is used as the loss function, the problem to solve is the standard support vector machine (Vapnik 1998), which is studied in the next chapter.

8.2.2 Kernels

A kernel function converts information on a pair of subjects into a quantitative measure representing their similarity with the requirement that the function must create a symmetric positive semi-definite (psd) matrix when applied to any subset of subjects. The psd requirement ensures a statistical foundation for using the kernel in penalized regression models. From a statistical perspective, the kernel matrix can be viewed as a covariance matrix, and we later show how this aids in the construction of kernels. Kernels are used to nonlinearly transform the input data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \boldsymbol{X}$ into a high-dimensional feature space. Next, we provide a definition of kernel function.

Kernel function. Kernel function K is a "similarity" function that corresponds to an inner product in some expanded feature space that for all $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$ satisfies

$$K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \varphi(\boldsymbol{x}_i)^{\mathrm{T}} \varphi\left(\boldsymbol{x}_j\right),$$

where φ is a mapping (transformation) from $\boldsymbol{X}$ to an (inner product) feature space F, $\varphi : \mathbf{x} \rightarrow \varphi(\mathbf{x})$. From this definition, we can see that the kernel has the following properties:

1. It is a symmetric function of its argument so that $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K(\boldsymbol{x}_j, \boldsymbol{x}_i)$.
2. A necessary and sufficient condition for a function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be a valid kernel (Shawe-Taylor and Cristianini 2004) is that the Gram matrix, also called kernel matrix $\mathbf{K}$, whose elements are given by $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, should be positive semi-definite for all possible choices of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \boldsymbol{X}$.
3. Kernels are all those functions $K(\boldsymbol{u}, \boldsymbol{v})$ that verify Mercer's theorem, that is, for which

$$\int_{\boldsymbol{u},\boldsymbol{v}} K(\boldsymbol{u}, \boldsymbol{v}) g(\boldsymbol{u}) g(\boldsymbol{v}) d\boldsymbol{u} d\boldsymbol{v} > 0$$

for all $g()$ square-integrable functions.

Mercer's theorem is an equivalent formulation of the finitely positive semi-definite property for vector spaces. The finitely positive semi-definite property suggests that kernel matrices form the core data structure for kernel methods technology. By manipulating kernel matrices, one can tune the corresponding embedding of the data in the kernel-defined feature space.

Next, we give an example of the utility of a kernel function and how this works. We assumed that we measured a sample of n plants with two independent variables (x_1, x_2) and one binary dependent variable (y), that is, $(x_{11}, x_{21}, y_1), \ldots, (x_{1n}, x_{2n}, y_n)$. Then we plotted the observed data in Fig. 8.1 (left panel), and since the response variable is binary, we used triangles for denoting diseased plants and crosses for non-diseased plants. The goal is to build a classifier for unseen data using the data given in Fig. 8.1 as the training set. It is not possible to create a linear decision

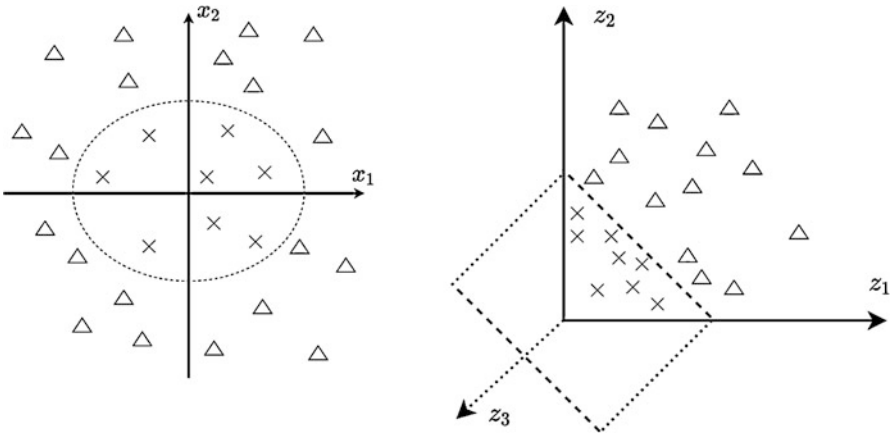

Fig. 8.1 Mapping of the two predictor (x_1, x_2) problems with binary dependent variables (y; crosses = 1 and triangles = 0) where the true decision boundary is an ellipse in predictor space (left panel) to a feature map via nonlinear mapping, $\varphi(\mathbf{x})$. Input space (left panel) and feature space (right panel)

boundary to separate both types of plants (diseased vs. non-diseased) since the true decision boundary is an ellipse in predictor space (Fig. 8.1, left panel). The job of a kernel consists of estimating this boundary by first transforming (mapping) the input information (predictors) via a nonlinear mapping function into a feature map, where the problem can be reduced to estimating a hyperplane (linear boundary) between the diseased and non-diseased plants. We mapped the input information (Fig. 8.1, left panel) to the feature space using the following nonlinear map $\varphi(\mathbf{x}) = \left(z_1 = x_1^2, z_2 = x_2^2, z_3 = \sqrt{2}x_1x_2\right)$ (Fig. 8.1, right panel) and the ellipse became a hyperplane that is parallel to the z_3 axis, which means that all points are plotted on the (z_1, z_2) plane. Therefore, in the feature space, the problem reduces to estimating a hyperplane from the mapped data points.

For this reason, in generalized kernel models, the choice of the kernel function (H) is of paramount importance since it defines the space of functions over which the search for f is performed, and because Hilbert spaces are normed spaces (Akhiezer and Glazman 1963). As mentioned above, by choosing H one automatically defines the reproducing kernel ($\boldsymbol{K}$) which should be at least a psd matrix (de los Campos et al. 2010). There are two main properties that are required for the successful implementation of a kernel function. First, it should capture as precisely as possible the measure of similarity to the particular task and domain, and second, its construction should require significantly less computational resources than would be needed for an explicit evaluation of the corresponding feature mapping, φ.

We will call the original input information ($\boldsymbol{X}$) the input space. Then, with the kernel approach, we define a function for each pair of elements (columns) in this space $\boldsymbol{X}$ that corresponds to a real value. The transformed feature information with the kernel function is called mapped feature space.

8.2.3 Kernel Trick

By kernel trick we mean the use of kernel functions to operate in a high-dimensional space, *implicit* feature space, without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This means that the kernel trick allows you to perform algebraic operations in the transformed data space efficiently and without knowing the transformation φ. For this reason, the kernel trick is a computational trick used to compute inner products in higher dimensional spaces at a low cost. Thus, in principle, any statistical machine learning technique for data in $\boldsymbol{X} \subset \mathbb{R}^n$ that can be formulated in a computational algorithm in terms of dot products can be generalized to the transformed data using the kernel trick. Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

To better understand the kernel trick, we provide an example. Assume that we measure two independent variables (x_1, x_2) in four individuals. In matrix notation, the information of the independent variables (input information) is equal to

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix}$$

Also, assume we will build a polynomial kernel of degree 2, with

$$\varphi(\mathbf{x}_i)^{\mathrm{T}} = \left(z_1 = x_1^2, z_2 = x_2^2, z_3 = \sqrt{2}x_1x_2\right).$$

Therefore, for building the Gram matrix (kernel matrix), we need to compute

$$\boldsymbol{K} = \begin{bmatrix} \varphi(\mathbf{x}_1)^{\mathrm{T}}\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_1)^{\mathrm{T}}\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_1)^{\mathrm{T}}\varphi(\mathbf{x}_3) & \varphi(\mathbf{x}_1)^{\mathrm{T}}\varphi(\mathbf{x}_4) \\ \varphi(\mathbf{x}_2)^{\mathrm{T}}\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_2)^{\mathrm{T}}\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_2)^{\mathrm{T}}\varphi(\mathbf{x}_3) & \varphi(\mathbf{x}_2)^{\mathrm{T}}\varphi(\mathbf{x}_4) \\ \varphi(\mathbf{x}_3)^{\mathrm{T}}\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_3)^{\mathrm{T}}\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_3)^{\mathrm{T}}\varphi(\mathbf{x}_3) & \varphi(\mathbf{x}_3)^{\mathrm{T}}\varphi(\mathbf{x}_4) \\ \varphi(\mathbf{x}_4)^{\mathrm{T}}\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_4)^{\mathrm{T}}\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_4)^{\mathrm{T}}\varphi(\mathbf{x}_3) & \varphi(\mathbf{x}_4)^{\mathrm{T}}\varphi(\mathbf{x}_4) \end{bmatrix}$$

This means that we need to compute each coordinate (cell of $\boldsymbol{K}$) with $\varphi(\mathbf{x}_i)^{\mathrm{T}}\varphi(\mathbf{x}_j)$, with $i, j = 1, 2, 3, 4$. Note that

$$\varphi(\mathbf{x}_i)^{\mathrm{T}}\varphi(\mathbf{x}_j) = \left(x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}\right)\begin{bmatrix} x_{j1}^2 \\ x_{j2}^2 \\ \sqrt{2}x_{j1}x_{j2} \end{bmatrix} = x_{i1}^2x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2x_{j2}^2$$

$$= \left(x_{i1}x_{j1} + x_{i2}x_{j2}\right)^2.$$

Therefore,

$$\boldsymbol{K} = \begin{bmatrix} \left(x_{11}^2 + x_{12}^2\right)^2 & \left(x_{11}x_{21} + x_{12}x_{22}\right)^2 & \left(x_{11}x_{31} + x_{12}x_{32}\right)^2 & \left(x_{11}x_{41} + x_{12}x_{42}\right)^2 \\ \left(x_{21}x_{11} + x_{22}x_{12}\right)^2 & \left(x_{21}^2 + x_{22}^2\right)^2 & \left(x_{21}x_{31} + x_{22}x_{32}\right)^2 & \left(x_{21}x_{41} + x_{22}x_{42}\right)^2 \\ \left(x_{31}x_{11} + x_{32}x_{12}\right)^2 & \left(x_{31}x_{21} + x_{32}x_{22}\right)^2 & \left(x_{31}^2 + x_{32}^2\right)^2 & \left(x_{31}x_{41} + x_{32}x_{42}\right)^2 \\ \left(x_{41}x_{11} + x_{42}x_{12}\right)^2 & \left(x_{41}x_{21} + x_{42}x_{22}\right)^2 & \left(x_{41}x_{31} + x_{42}x_{32}\right)^2 & \left(x_{41}^2 + x_{42}^2\right)^2 \end{bmatrix}$$

To compute $\boldsymbol{K}$ we calculated each coordinate using $\varphi(\mathbf{x}_i)^{\mathrm{T}}\varphi(\mathbf{x}_j)$. However, note that

$$\varphi(\mathbf{x}_i)^{\mathrm{T}}\varphi(\mathbf{x}_j) = \left(x_{i1}x_{j1} + x_{i2}x_{j2}\right)^2 = \left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + 0\right)^2,$$

where $\boldsymbol{x}_i^{\mathrm{T}} = [x_{i1}, x_{i2}]$ and $\boldsymbol{x}_j = \begin{bmatrix} x_{j1} \\ x_{j2} \end{bmatrix}$.

Hence, the function

$$K\left(\mathbf{x}_j, \mathbf{x}_j\right) = \left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + 0\right)^2$$

corresponds to a polynomial kernel of degree $d = 2$, and constant $a = 0$, with F, its corresponding feature space. This means that we can compute the inner product between the projections of two points into the feature space without explicitly evaluating the coordinates. In other words, the kernel trick means that we can compute each element (coordinate) of the kernel matrix $\mathbf{K}$, without any knowledge of the true nature of $\varphi(\mathbf{x}_i)$; we only need to know the kernel function $K(\mathbf{x}_j, \mathbf{x}_j)$. This means that the kernel function is a key ingredient for implementing kernel methods in statistical machine learning.

Next, we provide another simple example also using the polynomial kernel of degree 2, with the same two independent variables (x_1, x_2) but with a constant value $a = 1$, that is, $K(\mathbf{x}_j, \mathbf{x}_j) = \left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + 1\right)^2$. According to the kernel trick, this means that we do not need knowledge of $\varphi(\mathbf{x}_i)$ to compute all coordinates of the matrix of kernel $\mathbf{K}$, since each coordinate will take values of

$$
\begin{aligned}
K(\mathbf{x}_j, \mathbf{x}_j) &= \left(\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + 1\right)^2 = \left(x_{i1}x_{j1} + x_{i2}x_{j2} + 1\right)^2 \\
&= \left(x_{i1}x_{j1} + x_{i2}x_{j2}\right)^2 + 2\left(x_{i1}x_{j1} + x_{i2}x_{j2}\right) + 1 \\
&= x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 + 2\left(x_{i1}x_{j1} + x_{i2}x_{j2}\right) + 1.
\end{aligned}
$$

Therefore, the **K** matrix is

$$
= \begin{bmatrix}
\left(x_{11}^2 + x_{12}^2 + 1\right)^2 & \left(x_{11}x_{21} + x_{12}x_{22} + 1\right)^2 & \left(x_{11}x_{31} + x_{12}x_{32} + 1\right)^2 & \left(x_{11}x_{41} + x_{12}x_{42} + 1\right)^2 \\
\left(x_{21}x_{11} + x_{22}x_{12} + 1\right)^2 & \left(x_{21}^2 + x_{22}^2 + 1\right)^2 & \left(x_{21}x_{31} + x_{22}x_{32} + 1\right)^2 & \left(x_{21}x_{41} + x_{22}x_{42} + 1\right)^2 \\
\left(x_{31}x_{11} + x_{32}x_{12} + 1\right)^2 & \left(x_{31}x_{21} + x_{32}x_{22} + 1\right)^2 & \left(x_{31}^2 + x_{32}^2 + 1\right)^2 & \left(x_{31}x_{41} + x_{32}x_{42} + 1\right)^2 \\
\left(x_{41}x_{11} + x_{42}x_{12} + 1\right)^2 & \left(x_{41}x_{21} + x_{42}x_{22} + 1\right)^2 & \left(x_{41}x_{31} + x_{42}x_{32} + 1\right)^2 & \left(x_{41}^2 + x_{42}^2 + 1\right)^2
\end{bmatrix}
$$

This implies that we computed each coordinate of **K** without first computing $\varphi(\mathbf{x}_i)$. This trick is really useful since for computing each coordinate of **K** in this example, we only performed dot products with vectors of size two, in the original dimension of the input information, and not dot products of vectors of dimension $\binom{2+2}{2} = 6$, which is the dimension, in this example, of $\varphi(\mathbf{x}_i)^{\mathrm{T}} = \left[x_{i1}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i1}x_{i2}, \sqrt{2}x_{i2}, x_{i2}^2, 1\right]$. Therefore, this trick facilitates the computation of **K** since it requires less computation resources. The utility of the trick is better appreciated in a large dimensional setting. For example, assume that the input information of each individual ($\mathbf{x}_i$) contains 784 independent variables; this means that to compute matrix **K** we need to compute, for each coordinate, only dot products of vectors of dimension 784 and not of dimension $\binom{784+2}{2} = 308,505$, which is the dimension of $\varphi(\mathbf{x}_i)^{\mathrm{T}}$ for the same polynomial kernel with degree 2. For this reason, kernel methods are well suited for handling a massive amount of information, because the computational burden can be proportional to the number of data points rather than to the number of predictor variables (e.g., markers in the context of genomic prediction). This is particularly true if a common weight is assigned to each marker (Morota et al. 2013).

In simple terms, the *kernel trick* makes it possible to perform a transformation from the input data space to a higher dimensional feature space, where the transformed data can be analyzed with conventional linear models and the problem becomes tractable. However, the result highly depends on the considered transformation. If the kernel function is not appropriate for the problem, or the kernel parameters are badly set, the fitted model can be of poor quality. Due to this, special care must be taken when selecting both the kernel function and the kernel parameters to obtain good results.

The kernel trick allows an efficient search in a higher dimensional space, while the related estimation problems are often cast as convex optimization problems that can be solved by many established algorithms and packages. Kernel methods can be

applied to all data analysis algorithms whose inputs can be expressed in terms of dot products. If the data in the original space cannot be analyzed satisfactorily with conventional statistical machine learning techniques, the strategy to extend it to nonlinear models using kernel methods is based on the apparently paradoxical idea of transforming the data, by means of a nonlinear function, toward a space with a greater dimension than the space where the data are located and applying any statistical machine learning algorithm to the transformed data.

Therefore, in general terms, the kernelization of an algorithm consists of its reformulation, so that the determination of a pattern or linear regularity in the data can be carried out exclusively from the information collected in the scalar products calculated for all the pairs of elements in the space. Kernel functions are characterized by the property that all finite kernel matrices are positive semi-definite.

8.2.4 Popular Kernel Functions

Next, we provide the most popular kernel methods in statistical machine learning.

Linear Kernel This kernel is defined as $K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$. For example,

$$K(\boldsymbol{x}, \boldsymbol{z}) = (x_1, x_2) \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = x_1 z_1 + x_2 z_2 = \varphi(\boldsymbol{x})^{\mathrm{T}} \varphi(\boldsymbol{z}).$$

Next, we provide an *R* function for calculating this kernel that can be used for both single-attribute value vectors and for the whole data set:

```
K.linear=function(x1, x2=x1) {as.matrix(x1)%*%t(as.matrix(x2)) }
```

Next, we simulate a matrix data set:

```
set.seed(3)
X=matrix(round(rnorm(16,2,0.2),2),ncol=8)
X
```

that gives as output:

```
> set.seed(3)
> X=matrix(round(rnorm(16,2,0.2),2),ncol=8)
> X
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 1.81 2.05 2.04 2.02 1.76 1.85 1.86 2.03
[2,] 1.94 1.77 2.01 2.22 2.25 1.77 2.05 1.94
```

For individual features in pairs of individuals, this function is used as

```
> K.linear(X[1,1:4],X[2,1:4])
     [,1]   [,2]   [,3]   [,4]
[1,] 3.5114 3.2037 3.6381 4.0182
[2,] 3.9770 3.6285 4.1205 4.5510
[3,] 3.9576 3.6108 4.1004 4.5288
[4,] 3.9188 3.5754 4.0602 4.4844
```

while for the full set of features, it can be used as

```
> K.linear(X)
     [,1]    [,2]
[1,] 29.8212 30.7104
[2,] 30.7104 32.0265
```

This kernel does not overcome the linearity limitation of linear classification and linear regression models in any way since it leaves the original representation unchanged. It is important to point out that linear kernels (such as linear regression, linear support vector machines, and linear support vector regression algorithms) are special cases of more sophisticated kernel-based algorithms.

Polynomial Kernel As mentioned above, this kernel is defined as $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\gamma \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + a)^d$, where a is a real scalar and d is a positive integer, and where $\gamma > 0$, $a \geq 0$, and $d > 0$ are parameters. This kernel family makes it possible to easily control the enhanced representation size and degree of nonlinearity by adjusting the d parameter. Positive a can be used to adjust the relative impact of higher order and lower order terms in the resulting polynomial representation. For example, when $\gamma = 1$, $a = 0$, and $d = 2$, we have

$$K(\boldsymbol{x}, \boldsymbol{z}) = \left((x_1, x_2) \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right)^2 = (x_1 z_1 + x_2 z_2)^2 =$$

$$x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 = \left[x_1^2, \sqrt{2} x_1 x_2, x_2^2 \right] \begin{bmatrix} z_1^2 \\ \sqrt{2} z_1 z_2 \\ z_2^2 \end{bmatrix} = \varphi(\boldsymbol{x})^{\mathrm{T}} \varphi(\boldsymbol{z}).$$

However, when $\gamma = 1$, $a = 1$, and $d = 2$, we have

$$K(\boldsymbol{x}, \boldsymbol{z}) = \left((x_1, x_2) \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} + 1 \right)^2 = (x_1 z_1 + x_2 z_2 + 1)^2 =$$

$$1 + 2x_1z_1 + 2x_2z_2 + x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1x_2z_2$$

$$= \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2\right] \begin{bmatrix} 1 \\ \sqrt{2}z_1 \\ \sqrt{2}z_2 \\ z_1^2 \\ \sqrt{2}z_1z_2 \\ z_2^2 \end{bmatrix} = \varphi(\boldsymbol{x})^{\mathrm{T}}\varphi(\boldsymbol{z}).$$

This demonstrates that increasing a increases the coefficients of lower order terms. The dimension of the feature space for the polynomial kernel is equal to $\binom{p+d}{d}$. For example, for an input vector of dimension $p = 10$ and polynomial with degree $d = 3$, the dimension for this polynomial kernel is equal to $\binom{10+3}{3}$ =286, while if $p = 1000$ and $d = 3$, the dimension for this polynomial kernel is equal to $\binom{1000+3}{3}$ =167,668,501. Although convenient to control and easy to understand, the polynomial kernel family may be insufficient to adequately represent more complex relationships.

The R code for calculating this kernel is given next:

```
K.polynomial=function(x1, x2=x1, gamma=1, b=0, d=3){
 (gamma*(as.matrix(x1)%*%t(x2))+b)^d}
```

Now this function can be used as

```
> K.polynomial(X[1,1:4],X[2,1:4])
     [,1]     [,2]     [,3]     [,4]
[1,] 43.29532 32.88180 48.15306 64.87758
[2,] 62.90234 47.77288 69.95999 94.25850
[3,] 61.98630 47.07716 68.94117 92.88582
[4,] 60.18099 45.70607 66.93331 90.18058
```

But for the full set of features, it can be used as

```
> K.polynomial(X)
     [,1]     [,2]
[1,] 26520.11 28963.86
[2,] 28963.86 32849.48
```

Sigmoidal Kernel This kernel is defined as $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tan h(\gamma \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + b)$, where tan$h$ is the hyperbolic tangent defined as $\tan h(z) = \sin h(z)/\cos h(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$. This function is widely used as the activation function for artificial neural networks and deep learning models, and hence has also become popular for kernel methods. If used with properly adjusted parameters, it can represent complex nonlinear relationships. In some parameter settings, it actually becomes similar to the radial kernel (Lin and Lin 2003) described below. However, the sigmoid function may not be positive definite for some parameters, and therefore may not actually represent a valid kernel (Lin and Lin 2003).

Next, we provide an R code for calculating this kernel:

```
K.sigmoid=function(x1,x2=x1, gamma=0.1, b=0)
{ tanh(gamma*(as.matrix(x1)%*%t(x2))+b) }
```

This function is used as

```
> K.sigmoid(X[1,1:4],X[2,1:4])
     [,1]      [,2]      [,3]      [,4]
[1,] 0.3373862 0.3098414 0.3485656 0.3815051
[2,] 0.3779793 0.3477219 0.3902119 0.4260822
[3,] 0.3763152 0.3461650 0.3885066 0.4242635
[4,] 0.3729798 0.3430454 0.3850881 0.4206158
```

For the full set of features, it is used as

```
> K.sigmoid(X)
     [,1]      [,2]
[1,] 0.9948752 0.9957083
[2,] 0.9957083 0.9966999
```

Gaussian Kernel This kernel, also known as the radial basis function kernel, depends on the Euclidean distance between the original attribute value vectors (i.e., the Euclidean norm of their difference) rather than on their dot product, $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2} = e^{-\gamma \left[\boldsymbol{x}_i^T \boldsymbol{x}_i - 2\boldsymbol{x}_i^T \boldsymbol{x}_i + \boldsymbol{x}_j^T \boldsymbol{x}_j \right]}$, where γ is a positive real scalar. It is known that the feature vector φ that corresponds to the Gaussian kernel is actually infinitely dimensional (Lin and Lin 2003). Therefore, without the kernel trick, the solution cannot be computed explicitly. This type of kernel tends to be particularly popular, but it is sensitive to the choice of the γ parameter and may be prone to overfitting.

The R code for calculating this kernel is given next:

```
l2norm=function(x){sqrt(sum(x^2))}
K.radial=function(x1,x2=x1, gamma=1){
 exp(-gamma*outer(1:nrow(x1 <- as.matrix(x1)), 1:ncol(x2 <- t(x2)),
          Vectorize(function(i, j) l2norm(x1[i,]-x2[,j]) ^2)))}
```

This function is used as

```
> K.radial(X[1,1:4],X[2,1:4])
     [,1]      [,2]      [,3]      [,4]
[1,] 0.9832420 0.9984013 0.9607894 0.8452693
[2,] 0.9879729 0.9245945 0.9984013 0.9715136
[3,] 0.9900498 0.9296938 0.9991004 0.9681193
[4,] 0.9936204 0.9394131 0.9999000 0.9607894
```

while for the full set of features, it can be used as

```
> K.radial(X)
     [,1]      [,2]
[1,] 1.0000000 0.6525288
[2,] 0.6525288 1.0000000
```

The parameter γ controls the flexibility of the Gaussian kernel in a similar way as the degree d in the polynomial kernel. Large values of γ correspond to large values of d since, for example, they allow classifiers to fit any labels, hence risking overfitting. In such cases, the kernel matrix becomes close to the identity matrix. On the other hand, small values of γ gradually reduce the kernel to a constant function, making it impossible to learn any nontrivial classifier. The feature space has infinite dimensions for every value of γ, but for large values, the weight decays very fast on the higher order features. In other words, although the rank of the kernel matrix is full, for all practical purposes, the points lie in a low-dimensional subspace of the feature space.

Exponential Kernel This kernel is defined as $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|}$, which is quite similar to the Gaussian kernel function.

The R code is given below:

```
K.exponential=function(x1,x2=x1, gamma=1){
 exp(-gamma*outer(1:nrow(x1 <- as.matrix(x1)), 1:ncol(x2 <- t(x2)),
         Vectorize(function(i, j) l2norm(x1[i,]-x2[,j]))))}
```

For individual features in pairs of individuals, it can be used as

```
> K.exponential(X[1,1:4],X[2,1:4])
     [,1]      [,2]      [,3]      [,4]
[1,] 0.8780954 0.9607894 0.8187308 0.6636503
[2,] 0.8958341 0.7557837 0.9607894 0.8436648
[3,] 0.9048374 0.7633795 0.9704455 0.8352702
[4,] 0.9231163 0.7788008 0.9900498 0.8187308
```

while for full set of features, it can be used as

```
> K.exponential(X)
     [,1]      [,2]
[1,] 1.0000000 0.5202864
[2,] 0.5202864 1.0000000
```

Arc-Cosine Kernel (AK) For AK, an important component is the angle between two vectors computed from inputs $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ as

$$\theta_{i,j} = \cos^{-1}\left(\frac{\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}\right),$$

where $\|\boldsymbol{x}_i\|$ is the norm of observation i. The following kernel is positive semi-definite and related to an ANN with a single hidden layer and the ramp activation function (Cho and Saul 2009).

$$\mathrm{AK}^1(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{\pi}\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\| \, J\left(\theta_{i,j}\right), \tag{8.4}$$

where π is the pi constant and $J(\theta_{i,j}) = [\sin(\theta_{i,j}) + (\pi - \theta_{i,j})\cos(\theta_{i,j})]$. Equation (8.4) gives a symmetric positive semi-definite matrix (AK^1) preserving the norm of the entries such that $\mathrm{AK}(\boldsymbol{x}_i, \boldsymbol{x}_i) = \|\boldsymbol{x}_i\|^2$ and $\mathrm{AK}(\boldsymbol{x}_i, -\boldsymbol{x}_i) = 0$ and models nonlinear relationships.

Note that the diagonals of the AK matrix are not homogeneous and express heterogeneous variances of the genetic value $\boldsymbol{u}$; this is different from the Gaussian kernel matrix, with a diagonal that expresses homogeneous variances. This property could be a theoretical advantage of AK when modeling interrelationships between individuals.

In order to emulate the performance of an ANN with more than one hidden layer (l), Cho and Saul (2009) proposed a recursive relationship of repeating l times the interior product:

$$\mathrm{AK}^{(l+1)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)=\frac{1}{\pi}\left[\mathrm{AK}^{(l)}(\boldsymbol{x}_i,\boldsymbol{x}_i)\mathrm{AK}^{(l)}\left(\boldsymbol{x}_j,\boldsymbol{x}_j\right)\right]^{\frac{1}{2}}J\left(\theta_{i,j}^{(l)}\right), \tag{8.5}$$

where $\theta_{i,j}^{(l)}=\cos^{-1}\left\{\mathrm{AK}^{(l)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)\left[\mathrm{AK}^{(l)}(\boldsymbol{x}_i,\boldsymbol{x}_i)\mathrm{AK}^{(l)}\left(\boldsymbol{x}_j,\boldsymbol{x}_j\right)\right]^{-\frac{1}{2}}\right\}$.

Thus, computing $\mathrm{AK}^{(l+1)}$ at level (layer) $l+1$ is done from the previous layer $\mathrm{AK}^{(l)}$. Computing a bandwidth (the smoothing parameter that controls variance and bias in the output, e.g., the γ parameter in the Gaussian kernel) is not necessary, and the only computational effort required is to compute the number of hidden layers. Cuevas et al. (2019) described a maximum marginal likelihood method used to select the number of hidden layers (l) for the AK kernel. It is important to point out that this kernel method is like a deep neural network since it allows using more than one hidden layer.

The R code for the AK kernel with one hidden layer is given below:

```
K.AK1_Final<-function(x1,x2){
 n1<-nrow(x1)
 n2<-nrow(x2)
 x1tx2<-x1%*%t(x2)
 norm1<-sqrt(apply(x1,1,function(x) crossprod(x)))
 norm2<-sqrt(apply(x2,1,function(x) crossprod(x)))
 costheta = diag(1/norm1)%*%x1tx2%*%diag(1/norm2)
 costheta[which(abs(costheta)>1,arr.ind = TRUE)] = 1
 theta<-acos(costheta)
 normx1x2<-norm1%*%t(norm2)
 J = (sin(theta)+(pi-theta)*cos(theta))
 AK1 = 1/pi*normx1x2*J
 AK1<-AK1/median(AK1)
 colnames(AK1)<-rownames(x2)
 rownames(AK1)<-rownames(x1)
 return(AK1)
}
```

For the full set of features, it can be used as

```
> K.AK1_Final(x1=X,x2=X)
     [,1]    [,2]
[1,] 0.9709  1.000000
[2,] 1.0000  1.042699
```

Since the K.AK1_Final() kernel function is only useful for one hidden layer, for this reason, the next part of the code extends this to more than one hidden layer.

```
####Kernel Arc-Cosine with deep=4#####
diagAK_f<-function(dAK1)
{
 AKAK = dAK1^2
 costheta = dAK1*AKAK^(-1/2)
 costheta[which(costheta>1,arr.ind = TRUE)] = 1
```

```
 theta = acos(costheta)
 AKl  = (1/pi)*(AKAK^(1/2))*(sin(theta)+(pi-theta)*cos(theta))
 AKl
 AKl<-AKl/median(AKl)
}
AK_L_Final<-function(AK1,dAK1,nl){
 n1<-nrow(AK1)
 n2<-ncol(AK1)
 AKl1 = AK1
 for ( l in 1:nl){

  AKAK<-tcrossprod(dAK1,diag(AKl1))

  costheta<-AKl1*(AKAK^(-1/2))
  costheta[which(costheta>1,arr.ind = TRUE)] = 1
  theta <-acos(costheta)

  AKl<-(1/pi)*(AKAK^(1/2))*(sin(theta)+(pi-theta)*cos(theta))
  dAKl = diagAK_f(dAK1)
  AKl1 = AKl
  dAK1 = dAKl
 }
 AKl<-AKl/median(AKl)
 rownames(AKl)<-rownames(AK1)
 colnames(AKl)<-colnames(AK1)
 return(AKl)
}
```

Next, we illustrate how to use this kernel function for an AR kernel with four hidden layers:

```
> AK1=K.AK1_Final(x1=X,x2=X)
> AK_L_Final(AK1= AK1,dAK1=diag(AK1),nl=4)
     [,1]       [,2]
[1,] 0.9649746 1.000000
[2,] 1.0000000 1.036335
```

Hybrid Kernel We understand by hybrid kernels when two or more kernels are combined, since complex kernels can be created by simple operations (multiplication, addition, etc.) that combine simpler kernels. An example of a hybrid kernel can be obtained by multiplying the polynomial kernel and the Gaussian kernel. This kernel is defined as $\left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + a\right)^d e^{-\gamma\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|}$. However, other types of kernels can also be combined in the same fashion or with other basic operations, like kernel averaging, which is explained next.

Kernel Averaging Averaging is another way to create hybrid kernels, since kernel methods do not preclude the use of several kernels together (de los Campos et al. 2010). To illustrate the construction of these kernels, we assume that we have three

kernels $\boldsymbol{K}_1$, $\boldsymbol{K}_2$, and $\boldsymbol{K}_3$ that are distinct from each other. In this approach, the three kernels are "averaged" to form a new kernel $\boldsymbol{K} = \boldsymbol{K}_1 \frac{\sigma^2_{K_1}}{\sigma^2_K} + \boldsymbol{K}_2 \frac{\sigma^2_{K_2}}{\sigma^2_K} + \boldsymbol{K}_3 \frac{\sigma^2_{K_3}}{\sigma^2_K}$, where $\sigma^2_{K_1}, \sigma^2_{K_2}, \sigma^2_{K_3}$ are variance components attached to kernels $\boldsymbol{K}_1$, $\boldsymbol{K}_2$, and $\boldsymbol{K}_3$, respectively, and σ^2_K is the sum of the three variances. The ratios of the three variance components are tantamount to the relative contributions of the kernels. For instance, the kernels used can be three Gaussian kernels with different bandwidth parameter values, as employed in Tusell et al. (2013), or one can fit several parametric kernels jointly, e.g., the additive (**G**), dominance (**D**), and additive by dominance (**G#D**) kernels, as in Morota et al. (2014). While there are many possible choices of kernels, the kernel function can be estimated via maximum likelihood by recourse to the Matérn family of covariance functions (e.g., Ober et al. 2011) or by fitting several candidate kernels simultaneously through multiple kernel learning.

Hybrid kernels illustrate a general principle of how more complex kernels can be created from simpler ones in a number of different ways. Kernels can even be constructed that correspond to infinite-dimensional feature spaces at the cost of only a few extra operations in the kernel evaluations, like the Gaussian kernel which most often is good enough (Ober et al. 2011). There are many other kernels, however, the above-mentioned kernels are the most popular. For example, Morota et al. (2013) evaluated diffusion kernels for discrete inputs with animal and plant data, and compared these to the Gaussian kernel. Differences in predictive ability were minimal; this is fortunate because computing diffusion kernels is time-consuming.

The bandwidth parameter can be selected based on (1) a cross-validation procedure, (2) restricted maximum likelihood (Endelman 2011), and (3) an empirical Bayesian method such as the one proposed by Pérez-Elizalde et al. (2015). The optimal value of the bandwidth parameter is expected to change with many factors such as (a) distance function, (b) number of markers, allelic frequency, and coding of markers, all markers affecting the distribution of observed distances, and (c) genetic architecture of the trait, a factor affecting the expected prior correlation of genetic values (de los Campos et al. 2010).

As pointed out above, kernel methods only need information of the kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, assuming that this has been defined. For this reason, nonvectorial patterns $\mathbf{x}$ such as sequences, trees, and graphs can be handled. That is, kernel functions are not restricted to vectorial inputs: kernels can be designed for objects and structures as diverse as strings, graphs, text documents, sets, and graph nodes. It is important to point out that the kernel trick can be applied in unsupervised methods like cluster analysis, and dimensionality reduction methods like principal component analysis, independent component analysis, etc.

8.2.5 A Two Separate Step Process for Building Kernel Machines

The goal of this section is to emphasize that the building process of kernel machines consists of two general independent steps. The first one consists of calculating the Gram matrix (kernel matrix $\boldsymbol{K}$) using only the information of the independent variables (input). This means that in this process the user needs to define the type of kernel function that he (she) will use in such a way as to capture the hypothesized nonlinear patterns in the input data. Then in the second step, after the kernel is ready, we select the statistical machine learning algorithm that will be used for training the model using the dependent variable, the kernel built in the first step and other available covariates. These two separate steps for building kernel methods for prediction imply that we can use conventional linear statistical machine learning algorithms to accommodate a particular type of kernel function. The only important consideration when choosing the kernel is that it should be suitable for the data at hand. But, if you built the kernel, you can evaluate the performance of this kernel with many other statistical machine learning methods. This illustrates the two separate steps required for training predictive machines using kernel methods where any statistical machine learning method can be combined with any kernel function. It is important to point out that since many machine learning methods are only able to work with linear patterns, using the kernel trick allows you to build nonlinear versions of the linear algorithms, without the need to modify the original machine learning algorithm. The following sections show how the kernel trick works in some standard statistical machine learning models.

8.3 Kernel Methods for Gaussian Response Variables

When the response variable is Gaussian, the negative log-likelihood that needs to be used to minimize expression (8.3) belongs to a normal distribution and the expression (8.3) is reduced to

$$\underbrace{\min}_{\eta_0,\boldsymbol{\beta}}\left\{\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\eta_0-\boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta}\right)^2+\frac{\lambda}{2}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\beta}\right\}.$$

In matrix notation, the latter expression can be expressed as

$$\underbrace{\min}_{\eta_0,\boldsymbol{\beta}}\left\{\frac{1}{2}(\boldsymbol{y}^*-\boldsymbol{K}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y}^*-\boldsymbol{K}\boldsymbol{\beta})+\frac{\lambda}{2}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\beta}\right\},$$

where $\boldsymbol{y}^*=\boldsymbol{y}-\mathbf{1}\bar{y}$, using $\bar{y}$ as an estimator of the intercept (η_0). The first-order conditions to this problem are familiar to us (see Chap. 3) and are

$$\left[\boldsymbol{K}^{\mathrm{T}}\boldsymbol{K}+\lambda\boldsymbol{K}\right]\boldsymbol{\beta}=\boldsymbol{K}^{\mathrm{T}}\boldsymbol{y}^{*}$$

Further, since $\boldsymbol{K}=\boldsymbol{K}^{\mathrm{T}}$ and $\boldsymbol{K}^{-1}$ exist, pre-multiplication by $\boldsymbol{K}^{-1}$ yields

$$\boldsymbol{\beta}=\left[\boldsymbol{K}+\lambda\boldsymbol{I}\right]^{-\mathbf{1}}\boldsymbol{y}^{*},$$

where $\boldsymbol{I}$ is an identity matrix of dimension $n \times n$, and to estimate $\boldsymbol{\beta}$, λ must be known. It is important to point out that even in the context of large p and small n, the number of beta coefficients ($\boldsymbol{\beta}$) that need to be estimated is equal to n, which considerably reduces the computation resources in the estimation process. This solution to the beta coefficients obtained under Gaussian response variables is known as kernel Ridge regression in statistical machine learning, and was first obtained by Gianola et al. (2006) and Gianola and van Kaam (2008) in the context of a mixed effects model under a Bayesian treatment. The predicted values in the original scale of the response variables can be obtained as

$$\widehat{\boldsymbol{y}}=\mathbf{1}\overline{\mathrm{y}}+\boldsymbol{K}\widehat{\boldsymbol{\beta}}.$$

For a new observation with vector of inputs ($\boldsymbol{x}_{\mathrm{new}}$), the predictions are made using the following expression:

$$\widehat{\boldsymbol{y}}_{\mathrm{new}}=\overline{\mathrm{y}}+\sum_{i=1}^{n}\widehat{\beta}_{i}K(\boldsymbol{x}_{i},\boldsymbol{x}_{\mathrm{new}})$$

Next, we provide some examples of Gaussian response variables using different kernel methods.

Example 1 for continuous response variables. The data comprise family, marker, and phenotypic information of 599 lines that were evaluated for grain yield (GY) in four environments. Marker information consisted of 1447 Diversity Array Technology (DArT) markers, generated by Triticarte Pty. Ltd. (Canberra, Australia). Also, this data set contains the pedigree relationship matrix and is preloaded in the BGLR package with the name wheat. We named this data set the wheat599 data set. The GY measured in the four environments was used for single environment analysis using various kernel methods.

The first six observations for trait GY in the four environments (labeled 1, 2, 4, and 5) are given next.

```
> head(y)
       1          2           4          5
775   1.6716295 -1.72746986 -1.89028479  0.0509159
2166 -0.2527028  0.40952243  0.30938553 -1.7387588
2167  0.3418151 -0.64862633 -0.79955921 -1.0535691
2465  0.7854395  0.09394919  0.57046773  0.5517574
```

```
3881 0.9983176 -0.28248062 1.61868192 -0.1142848
3889 2.3360969 0.62647587 0.07353311 0.7195856
```

Also, next are given the first six observations for five standardized markers

```
> head(XF[,1:5])
     wPt.0538   wPt.8463  wPt.6348 wPt.9992  wPt.2838
[1,] -1.3598855 0.2672768 0.772228 0.4419075 0.439209
[2,]  0.7341284 0.2672768 0.772228 0.4419075 0.439209
[3,]  0.7341284 0.2672768 0.772228 0.4419075 0.439209
[4,] -1.3598855 0.2672768 0.772228 0.4419075 0.439209
[5,] -1.3598855 0.2672768 0.772228 0.4419075 0.439209
[6,]  0.7341284 0.2672768 0.772228 0.4419075 0.439209
```

Then with the code given in Appendix 1, that uses the wheat599 data set, the nine kernels explained above were illustrated. We implemented the kernel Ridge regression method using the library glmnet. The results of the nine kernels for GY in each of the four environments are given next.

Table 8.1 indicates that the best predictions were observed in the four environments under the Sigmoid kernel and the worst under the polynomial kernel.

8.4 Kernel Methods for Binary Response Variables

When the response variable is binary, instead of using the sum of squares loss function that was used before for continuous response variables, we now use the negative log-likelihood of the product of Bernoulli distributions, and the expression that needs to be minimized is given next:

$$\underbrace{\min}_{\eta_0,\boldsymbol{\beta}}\left\{\frac{1}{n}\sum_{i=1}^{n}\left(y_i\left(\eta_0+\boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta}\right)+\log\left[1+\exp\left(\eta_0+\boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta}\right)\right]\right)^2+\frac{\lambda}{2}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\beta}\right\}$$

Estimation of the parameters η_0 and $\boldsymbol{\beta}$ requires an iterative procedure, and gradient descent methods are used for their estimation, like those explained for logistic regression in Chaps. 3 and 7. Here, for the examples, we will use the glmnet package.

In Table 8.2, we can observe the prediction performance using the binary trait Height of the Data_Toy_EYT.R with nine kernels (linear, polynomial, sigmoid, Gaussian, exponential, AK1, AK2, AK3, and AK4). The Data_Toy_EYT data set contains 160 observations with 40 in each of the four environments that are present. The phenotypic information consists of a column for lines, another for environments and four corresponding to traits, two measured on a categorical scale, one continuous, and the last one binary. The data set also contains a genomic relationship matrix of the 40 lines that were evaluated in each of the four environments. Ten fold cross-

Table 8.1 Prediction performance in terms of the mean square error for GY in the four environments (Env) under nine kernel methods

Env	Linear	Polynomial	Sigmoid	Gaussian	Exponential	AK1	AK2	AK3	AK4
1	1.009	1.001	**0.762**	0.893	0.892	0.882	0.881	0.886	0.891
2	0.916	1.069	**0.772**	0.922	0.905	0.841	0.899	0.901	0.909
4	0.984	1.004	**0.859**	0.940	0.950	0.941	0.936	0.943	0.945
5	1.019	1.002	**0.814**	0.901	0.949	0.868	0.876	0.885	0.894

Table 8.2 Prediction performance in terms of the proportion of cases correctly classified (PCCC) with ten fold cross-validation for the binary trait Height of the Data_Toy_EYT.R data set with nine kernels

Fold	Linear	Polynomial	Sigmoid	Gaussian	Exponential	AK1	AK2	AK3	AK4
1	0.813	0.813	0.688	0.813	0.813	0.813	0.813	0.813	0.813
2	0.688	0.688	0.563	0.688	0.688	0.688	0.688	0.688	0.688
3	0.688	0.750	0.563	0.625	0.625	0.625	0.625	0.625	0.750
4	0.625	0.750	0.375	0.750	0.750	0.625	0.750	0.625	0.750
5	0.750	0.750	0.625	0.750	0.750	0.750	0.688	0.750	0.688
6	0.625	0.688	0.625	0.625	0.625	0.625	0.625	0.625	0.688
7	0.750	0.750	0.500	0.750	0.750	0.750	0.750	0.750	0.750
8	0.813	0.813	0.563	0.813	0.813	0.813	0.813	0.813	0.813
9	0.688	0.688	0.625	0.688	0.688	0.688	0.688	0.688	0.688
10	0.813	0.875	0.688	0.813	0.813	0.813	0.813	0.750	0.813
Average	0.725	0.756	0.581	0.731	0.731	0.719	0.725	0.713	0.744

validation was implemented and the worst performance in terms of the proportion of cases correctly classified (PCCC) was with the sigmoid kernel and the best under the polynomial and AK4 kernels. The R code for reproducing the results in Table 8.2 is given in Appendix 2.

8.5 Kernel Methods for Categorical Response Variables

For categorical response variables, the loss function is the negative log-likelihood of the product of multinomial distributions and the expression that needs to be minimized is given next:

$$\underbrace{\min}_{\eta_0,\boldsymbol{\beta}}\left\{-\frac{1}{n}\left(\sum_{i=1}^{n}\sum_{c=1}^{C}I_{\{y_i=c\}}\left(\eta_{0c}+\boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta}_c\right)-\sum_{i=1}^{n}\log\left[\sum_{l=1}^{C}\exp\left(\eta_{0l}+\boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\beta}_l\right)\right]\right)+\frac{\lambda}{2}\sum_{l=1}^{C}\boldsymbol{\beta}_l^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\beta}_l\right\}$$

The estimation process does not have an analytical solution, and gradient descent methods are used for the estimation of the required parameters. The optimization process is done with the same methods described in Chaps. 3 and 7 for categorical response variables. The following illustrative examples were implemented using the glmnet library.

Now the Data_Toy_EYT.R data set was used that was also used for illustrating kernels with binary response variables. The nine kernels were implemented but with the categorical response variable days to heading (DTHD). Again, the worst predictions occurred with the sigmoid kernel, but now the best predictions were achieved with the AK2 kernel (Table 8.3). The code given in Appendix 2 can be used for reproducing these results with two small modifications: (a) replace the response variable y2=Pheno$Height with y2=Pheno$DTHD and (b) in the specification of the model in glmnet, replace family='binomial' with family='multinomial'.

8.6 The Linear Mixed Model with Kernels

Under a linear mixed model (LMM) ($\boldsymbol{y} = \boldsymbol{C\theta} + \boldsymbol{K\beta} + \boldsymbol{e}$), every individual i is associated with a genotype vector $\boldsymbol{x}_i^{\mathrm{T}}$ and a covariate vector $\boldsymbol{c}_i^{\mathrm{T}}$ (e.g., gender, age, herd, race, environment, etc.). Given a sample of individuals with a genotyped variants matrix $\boldsymbol{X} = \left[\boldsymbol{x}_1^{\mathrm{T}}, \boldsymbol{x}_2^{\mathrm{T}}, \ldots, \boldsymbol{x}_n^{\mathrm{T}}\right]^{\mathrm{T}}$ and matrix of incidence nuisance variables $\boldsymbol{C} = \left[\boldsymbol{c}_1^{\mathrm{T}}, \boldsymbol{c}_2^{\mathrm{T}}, \ldots, \boldsymbol{c}_n^{\mathrm{T}}\right]^{\mathrm{T}}$, relating some effect ($\boldsymbol{\theta}$) to the phenotype vector $\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n]^{\boldsymbol{T}}$ that follows a multivariate normal distribution.

$$\boldsymbol{y}|\boldsymbol{X},\boldsymbol{C} \sim \boldsymbol{N}\left(\boldsymbol{C\theta}, \boldsymbol{K} + \boldsymbol{I}\sigma_e^2\right)$$

Table 8.3 Prediction performance in terms of the proportion of cases correctly classified (PCCC) with ten fold cross-validation for the categorical trait days to heading (DTHD) of the Data_Toy_EYT.R data set with nine kernels

Fold	Linear	Polynomial	Sigmoid	Gaussian	Exponential	AK1	AK2	AK3	AK4
1	0.813	0.813	0.688	0.813	0.813	0.813	0.813	0.813	0.813
2	0.813	0.813	0.625	0.813	0.813	0.813	0.813	0.813	0.813
3	0.688	0.688	0.750	0.688	0.688	0.688	0.688	0.688	0.688
4	0.875	0.813	0.750	0.875	0.875	0.875	0.875	0.875	0.875
5	0.688	0.625	0.625	0.688	0.625	0.625	0.688	0.688	0.625
6	0.750	0.750	0.563	0.750	0.750	0.750	0.750	0.750	0.750
7	0.688	0.750	0.563	0.688	0.688	0.688	0.688	0.688	0.688
8	0.875	0.875	0.875	0.875	0.875	0.875	0.875	0.875	0.875
9	0.813	0.813	0.688	0.813	0.813	0.813	0.813	0.813	0.813
10	0.625	0.688	0.500	0.625	0.688	0.625	0.688	0.625	0.625
Average	0.763	0.763	0.663	0.763	0.763	0.756	0.769	0.763	0.756

Here, $\boldsymbol{K}$ is a valid kernel encoding genotypic covariance, as long as it is positive semi-definite and, again, represents similarities between genotyped individuals. Now the nonparametric function is $\boldsymbol{f(X)} = \boldsymbol{K\beta}$ and the nonparametric coefficients, $\boldsymbol{\beta}$, and residuals can be assumed to be independently distributed as $\boldsymbol{\beta} \sim \boldsymbol{N}\left(\boldsymbol{0}, \boldsymbol{K}^{-1}\sigma_\beta^2\right)$ and $\boldsymbol{e} \sim \boldsymbol{N}\left(\boldsymbol{0}, \boldsymbol{I}\sigma_e^2\right)$. $\boldsymbol{\theta}$ is a vector of covariate coefficients (denoted as fixed effects), $\boldsymbol{I}$ is the $n \times n$ identity matrix, and σ_e^2 is the variance of the microenvironmental effects. Now under this LMM approach, the function to be minimized becomes

$$\underbrace{\min}_{\boldsymbol{\theta},\boldsymbol{\beta}} \quad J[\boldsymbol{\theta},\boldsymbol{\beta}|\lambda] = \underbrace{\min}_{\boldsymbol{\theta},\boldsymbol{\beta}} \left\{ \frac{1}{2\sigma_e^2}[\boldsymbol{y} - \boldsymbol{C\theta} - \boldsymbol{K\beta}]^{\mathrm{T}}[\boldsymbol{y} - \boldsymbol{C\theta} - \boldsymbol{K\beta}] + \frac{\lambda}{2}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{K\beta} \right\}.$$

After setting the gradient of $J(.)$ with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ simultaneously to zero (Mallick et al. 2005; Gianola et al. 2006; Gianola and van Kaam 2008), the RKHS regression estimating equations can be formulated in matrix form given σ_e^2 and λ as

$$\begin{bmatrix} \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C} & \boldsymbol{C}^{\mathrm{T}}\boldsymbol{K} \\ \boldsymbol{K}^{\mathrm{T}}\boldsymbol{C} & \boldsymbol{K}^{\mathrm{T}}\boldsymbol{K} + \lambda\boldsymbol{K}\sigma_e^2 \end{bmatrix} \begin{bmatrix} \widehat{\boldsymbol{\theta}} \\ \widehat{\boldsymbol{\beta}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}^{\mathrm{T}}\boldsymbol{y} \\ \boldsymbol{K}^{\mathrm{T}}\boldsymbol{y} \end{bmatrix} \tag{8.6}$$

Recall that $\boldsymbol{K}$ is symmetric, so $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{K} = \boldsymbol{K}^2$, and by multiplying the second system of (8.6) by $\boldsymbol{K}^{-1}$ (assuming the inverse exists), we obtain

$$\begin{bmatrix} \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C} & \boldsymbol{C}^{\mathrm{T}}\boldsymbol{K} \\ \boldsymbol{I}^{\mathrm{T}}\boldsymbol{C} & \boldsymbol{K} + \lambda\boldsymbol{I}\sigma_e^2 \end{bmatrix} \begin{bmatrix} \widehat{\boldsymbol{\theta}} \\ \widehat{\boldsymbol{\beta}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}^{\mathrm{T}}\boldsymbol{y} \\ \boldsymbol{y} \end{bmatrix} \tag{8.7}$$

This avoids inverting $\boldsymbol{K}$ and forming $\boldsymbol{K}^{\mathrm{T}}\boldsymbol{K}$. Note that the variance of the nonparametric coefficient $\sigma_\beta^2 = \lambda^{-1}$ may be interpreted as variation due to marked additive genomic variation.

The mixed model $\boldsymbol{y} = \boldsymbol{C\theta} + \boldsymbol{K\beta} + \boldsymbol{e}$ (reparametrization I) can be reparametrized as $\boldsymbol{y} = \boldsymbol{C\theta} + \boldsymbol{u} + \boldsymbol{e}$ (reparametrization II), where $\boldsymbol{u} = \boldsymbol{K\beta}$, but with $\boldsymbol{u}$ distributed as $\boldsymbol{u} \sim \boldsymbol{N}\left(\boldsymbol{0}, \boldsymbol{K}\sigma_u^2\right)$, and σ_u^2 is the additive variance due to lines. Both parametrizations produce the same solution since they are equivalent, with the following peculiarities. Parametrization I has two main advantages: (1) kernel matrix $\boldsymbol{K}$ does not need to be inverted. The inverse of kernel matrix $\boldsymbol{K}$ may be time-consuming or unfeasible if the number of genotyped individuals is large, because the matrix is too dense. Currently, there is the need to invert the matrix up to 100,000×100,000. (2) Genome-enabled prediction of breeding values for any t new genotyped individuals ($\widehat{\boldsymbol{u}}_{\mathrm{new}}$) without phenotype can be done using a simple matrix–vector product $\widehat{\boldsymbol{u}}_{\mathrm{new}} = \boldsymbol{K}_s\widehat{\boldsymbol{\beta}}$, where $\widehat{\boldsymbol{\beta}}$ are the n nonparametric coefficients estimated from the n individuals in the training set, $\boldsymbol{K}_s$ is a matrix of dimension $(t \times n)$ containing the genomic similarity values between the t new individuals whose direct genomic merits are to be predicted and the individuals in the training set. When $\boldsymbol{K}$ is the genomic relationship matrix calculated as suggested by VanRaden (2008), the kernel is linear, but when $\mathbf{K}$ is

calculated with nonlinear kernels such as the Gaussian, exponential, polynomial, arc-cosine, sigmoid, etc., the same model can be used to capture nonlinear patterns better. This means that with the mixed model equations given above, it is possible to implement any of the proposed kernels since the only difference between them is the transformation performed on the input information to obtain a particular kernel. This means that the genomic relationship matrix used in a method known as genomic BLUP (GBLUP) is replaced by a more general kernel matrix that creates similarities among individuals, even if genetically unrelated. However, for a particular data set, some kernels will perform better and others worse, since the performance of the kernels is data-dependent. In our context, a kernel is any smooth function **K** defining a covariance structure among individuals. Also, as mentioned above, we can mix many kernels using a weighted sum or product of them to create new kernels. In general, as mentioned many times in this chapter, there is enough empirical evidence that kernel methods outperform conventional regression methods that are only able to capture linear patterns (Tusell et al. 2013; Long et al. 2010; Morota et al. 2013, 2014).

The solution of the mixed model equations given in (8.6 and 8.7) can be obtained using the rrBLUP package (Endelman 2011). This package is not restricted only to linear kernels since it is also useful for estimating marker effects by Ridge regression, and BLUPs calculated based on an additive relationship matrix. In this package, variance components are estimated by either maximum likelihood (ML) or restricted maximum likelihood (REML; default) using the spectral decomposition algorithm of Kang et al. (2008). The *R* function returns the variance components, the maximized log-likelihood (LL), the ML estimate for $\boldsymbol{\theta}$, and the BLUP solution for $\boldsymbol{u}$.

The basic function for implementing the kernel methods using the rrBLUP package is given next:

```
mixed.solve(y=y, Z=Z, K=K, X=X, method="REML"),
```

where $\boldsymbol{y}$, $\boldsymbol{Z}$, $\boldsymbol{K}$, and $\boldsymbol{X}$ are the vector of response variables, the design matrix of random effects, the kernel matrix, and the design matrix of fixed effects, respectively. The estimation method by default is REML, but the ML method is also allowed. The kernel matrix is calculated before using the mixed.solve() function. It is important to point out that the function kinship.BLUP allows implementing three kernel methods directly [linear kernel (RR), Gaussian kernel (GAUSS), and Exponential kernel (EXP)] under a mixed model approach.

```
kinship.BLUP(y=y[trn], G.train=W[trn,], G.pred=W[tst,], X=X
[trn,], K.method="GAUSS", mixed.method="REML"),
```

where **y**[trn] contains the training part of the response variable, **W**[trn,] contains the markers corresponding to the training set, **W**[tst,] contains the testing set of marker data, **X**[trn,] contains the fixed effects corresponding to the training set, the K. method="GAUSS" specifies that the Gaussian kernel will be implemented, and finally, mixed.method="REML" specifies any of the two estimation methods

Table 8.4 Prediction performance in terms of mean square error (MSE) and Pearson's correlation (PC) for each fold for the wheat599 (Environment 4) data set under a mixed model and three kernels: linear, Gaussian, and Exponential. These kernels are the defaults programmed in the rrBLUP library

	Linear	Gaussian	Exponential	Linear	Gaussian	Exponential
Fold	MSE	MSE	MSE	PC	PC	PC
1	0.757	0.694	0.720	0.534	0.592	0.610
2	0.667	0.626	0.630	0.536	0.583	0.590
3	0.774	0.685	0.700	0.435	0.521	0.490
4	0.736	0.609	0.649	0.398	0.535	0.509
5	0.677	0.690	0.685	0.454	0.428	0.426
6	1.139	1.036	1.025	0.309	0.395	0.405
7	1.006	0.966	1.010	0.486	0.523	0.514
8	0.874	0.758	0.752	0.486	0.594	0.616
9	0.683	0.592	0.586	0.526	0.621	0.625
10	0.729	0.683	0.689	0.315	0.366	0.365
Average	0.804	0.734	0.745	0.448	0.516	0.515

REML or ML. As mentioned above, this kinship.BLUP() allows implementing three kernels: linear, Gaussian, and Exponential. Using the wheat599 data set used in the last examples, a ten fold cross-validation using the three default kernels was implemented.

Table 8.4 shows that under a mixed model approach using the default kernels [linear (RR), Gaussian (GAUSS), and Exponential (EXP)] available in the rrBLUP library, the best predictions were observed in Env4 of data set wheat599 with the Gaussian and Exponential kernels under both metrics. The R code for reproducing the results in Table 8.4 is given in Appendix 3.

Table 8.5 provides the results of nine kernels for the response variable of Env4. The building process was manual for the kernel matrices and the data set used for this example was the wheat599 data set. Results for each of the nine kernels are given for each fold and across the 10 folds. Table 8.5 shows that the best prediction performance was observed with the Gaussian kernel and the worst under the polynomial kernel. The R code for reproducing the results in Table 8.5 is given in Appendix 4.

8.7 Hyperparameter Tuning for Building the Kernels

Hand-tuning kernel functions can be time-consuming and requires expert knowledge. A tuned kernel can improve the trained model, if standard kernels are insufficient for achieving a good transformation. In this section, we illustrate how to tune kernels and we compare the standard kernel with a hand-tuned kernel. As pointed out in Chap. 4, one approach to tuning is to divide the data into a training set, a tuning set, and a testing set. The training set is for training the data, the tuning set is for

Table 8.5 Prediction performance in terms of mean square error (MSE) for each fold for the wheat599 (Environment 4) data set under a mixed model, manually building the kernels linear, polynomial, sigmoid, Gaussian, exponential, AK1, AK2, AK3, and AK4

Fold	Linear	Polynomial	Sigmoid	Gaussian	Exponential	AK1	AK2	AK3	AK4
1	0.763	0.848	0.772	0.688	0.691	0.714	0.694	0.693	0.693
2	0.667	0.737	0.664	0.632	0.630	0.645	0.630	0.627	0.626
3	0.780	0.817	0.799	0.701	0.725	0.728	0.713	0.714	0.716
4	0.736	0.672	0.756	0.596	0.655	0.670	0.639	0.634	0.631
5	0.682	0.727	0.703	0.722	0.695	0.681	0.701	0.707	0.710
6	1.139	1.077	1.176	1.015	1.051	1.094	1.057	1.047	1.040
7	1.012	1.078	1.023	0.987	0.986	0.977	0.969	0.971	0.975
8	0.875	0.927	0.899	0.758	0.756	0.805	0.767	0.761	0.758
9	0.684	0.742	0.708	0.590	0.602	0.630	0.608	0.606	0.606
10	0.730	0.696	0.729	0.677	0.701	0.725	0.705	0.700	0.697
Average	0.807	0.832	0.823	0.736	0.749	0.767	0.748	0.746	0.745

choosing the best hyperparameter combination, and the testing set is for evaluating the prediction performance with the best hyperparameters. However, when the data sets are small after selecting the best combination of hyperparameters, the training and tuning sets are joined into one data set and with this data set the model is refitted again with the best combination of hyperparameters, and finally, the prediction performance is evaluated with the testing set.

This conventional approach to tuning is illustrated next using the wheat599 data set. To illustrate how to choose the hyperparameter, we will work with the arc-cosine kernel where the hyperparameter to be tuned is the number of hidden layers, for which we used a grid of 10 values (1, 2,. . ., 9, 10). To be able to tune the number of hidden layers, first we divided the original data set into four mutually exclusive parts with four fold cross-validation. This is called the outer cross-validation strategy. Then three of these parts were used for training and the remaining for testing. A ten fold cross-validation was performed in each of the outer training sets; this is called inner cross-validation. Nine out of the ten formed the inner training set and the remaining the tuning set. Then for each of the outer folds, the grid was evaluated with 10 values in the grid for the number of hidden layers, with the inner ten fold cross-validation strategy and for each of the 10 tuning sets, the mean square error of prediction was computed for each of the 10 values of the hidden layers in the grid. Then the average mean square error of the 10 inner cross-validations (in each outer fold) was computed for each value in the grid; it was selected as the optimal number of hidden layers in the grid that provides the smallest MSE. Then the inner training and the tuning sets (inner testing) were joined together for refitting the model with the optimal number of hidden layers, and finally, the prediction performance also in terms of MSE was evaluated in the outer testing set. The average of the four values of the outer testing set is reported as the final prediction performance. Under this strategy, the optimal number of hidden layers is different for each fold.

Figure 8.2 shows that the optimal number of hidden layers is different in each fold. In folds 1 and 3 (Fig. 8.2a, c), the optimal number of hidden layers was equal to 3, in folds 3 and 4 (Fig. 8.2b, d), the optimal number of hidden layers was equal to 8. Finally, with these optimal values, the model was refitted with the information of the inner training + tuning set, and then for each fold, the mean square error (MSE) was calculated for each outer testing set; the MSEs were 0.6878 (fold 1), 0.6963 (fold 2), 0.9725 (fold 3), and 0.7212 (fold 4), with an MSE across folds equal to 0.7694. The R code for reproducing these results is given in Appendix 5.

8.8 Bayesian Kernel Methods

For a single environment, the model can be expressed as

$$\boldsymbol{y} = \mu\mathbf{1} + \boldsymbol{u} + \boldsymbol{e}, \tag{8.8}$$