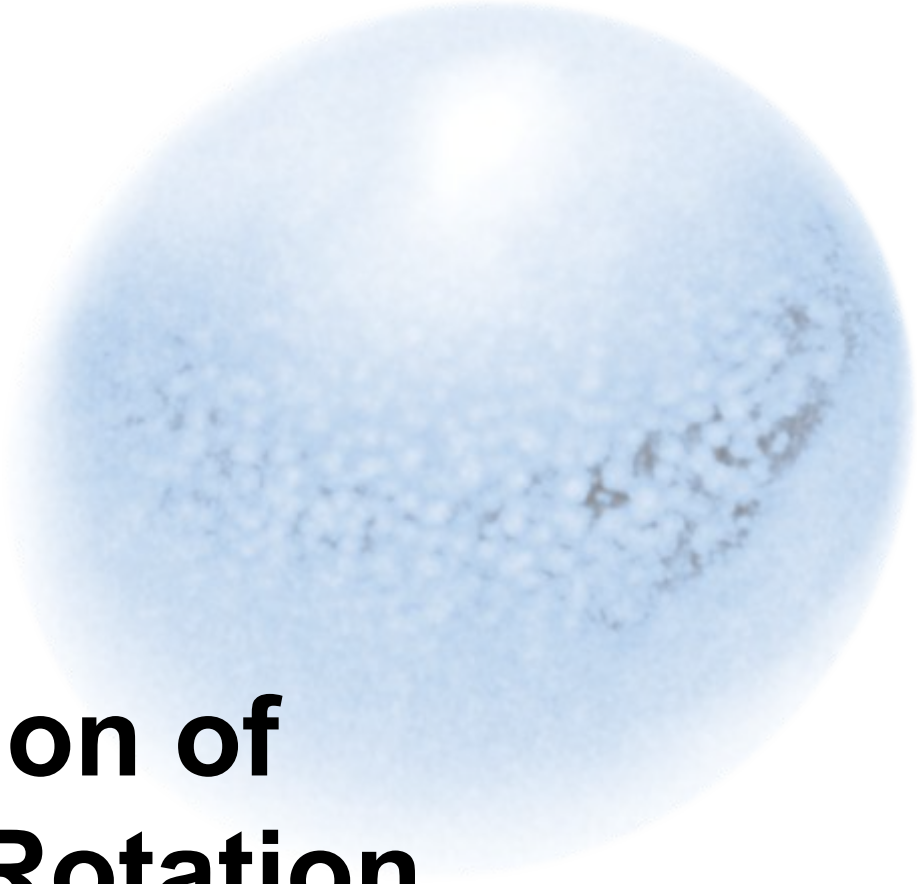

Prediction of Stellar Rotation Periods Using Regression Analysis

Nuno R. C. Gomes
Master's Degree in Data Science
Computer Science Department
2022

Supervisor
Luís Torgo, Associate Professor, Faculty of Sciences, University of Porto

Co-supervisor
Fabio Del Sordo, Institute of Space Sciences, Barcelona

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

The work in this dissertation is based on research carried out at the departments of *Computer Science* of *Faculdade de Ciências da Universidade do Porto*, Portugal, and *Dalhousie University*, Halifax – Nova Scotia, Canada. No part of this thesis has been submitted elsewhere for any other degree or qualification, and it is all the author's own work, unless otherwise stated in the text.

The image on the cover was created with the **DALL-E 2 AI system**, using the text description "surface rotating star with sunspots". It was further refined using the **Gimp – GNU Image Manipulation Program** software.

To my Nazanin.

Declaração de Honra

Eu, Nuno Renato Coelho Gomes, inscrito no Mestrado em Ciência de Dados da Faculdade de Ciências da Universidade do Porto, declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U. Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Nuno Gomes

Porto, 14 de novembro de 2022

Acknowledgements

I hereby acknowledge and express my deep gratitude to both my supervisor and co-supervisor, Luís Torgo and Fabio Del Sordo, for their guidance, advice, and support. This project would not have been possible without them.

I would also like to gratefully thank my teachers in the departments of Computer Science and Mathematics of FCUP, for widening my horizons, for providing me with essential researching tools, and for inspiring me throughout my curricular path.

A warm thanks to my friends and family, for their unconditional support.

Nuno Gomes
Porto, September 2022

Abstract

The estimation of rotation periods of stars is a key problem in stellar astrophysics. Reliable measurements of thousands of stellar rotation periods are fundamental for the study of the structure and evolution of stars, as well as of the Galaxy, to understand interactions between stars and their environment, and for the characterisation of planetary systems. Given the large amount of data available from ground-based and space telescopes, there is nowadays a growing concern to find reliable methods that allow one to quickly and automatically estimate stellar rotation periods accurately and with precision. This work is dedicated to the development of an approach to tackle this problem. We focused in building robust machine learning models to predict surface stellar rotation periods from structured data sets, built from the *Kepler* catalogue of K and M stars.

In order to make this thesis self-contained, we start by covering essential background in statistics, statistical learning, machine learning, and astrophysics. We then describe the materials and the methodology. We analyse the variables at hand, investigating the relationships between the independent variables and the response. We group the features according to their nature and methods with which they were obtained, and create eleven data sets using them. We then formalise the problem, proposing random forests and extreme gradient boosting approaches to create regression models that can help us meeting our research goals. We describe the sets of experiments we carried out using the random forests and gradient boosting frameworks, and present the results obtained with them. The assessment strategy relies on comparing goodness of fit metrics, such as the adjusted coefficient of determination, and the mean absolute value of the relative error. Our models demonstrate comparable predictive performance to other similar models published recently, while being reliable and computationally "cheap". Moreover, our approach comes the additional advantage of using data sets with fewer predictors, and, especially, we do not use rotation periods obtained with classical methods as input variables. We indicate the sets of most important variables to build solid machine learning models that can be used to automate the process of predicting the rotation period of K and M stars. Based on the results obtained with this study, we conclude that the best models obtained with the proposed methodology are competitive when compared with state of the art approaches.

Keywords: gradient boosting, machine learning, random forest, regression, stellar parameters, stellar rotation period, stellar structure.

Resumo

A determinação de períodos de rotação de estrelas é um problema chave da astrofísica estelar. Efetuar-se medições fiáveis de milhares de períodos de rotação estelar é fundamental para o estudo da estrutura e evolução das estrelas, bem como da Galáxia, para compreender as interações entre as estrelas e o seu meio ambiente, e para a caracterização de sistemas planetários. Dada a grande quantidade de dados disponíveis a partir de telescópios terrestres e espaciais, existe atualmente uma preocupação crescente em encontrar métodos fiáveis que permitam estimar rápida e automaticamente os períodos de rotação estelar com precisão. Este trabalho é dedicado ao desenvolvimento de uma abordagem para enfrentar este problema. Concentrámo-nos na construção de modelos robustos de aprendizagem de máquinas para prever períodos de rotação estelar a partir de conjuntos de dados estruturados, construídos a partir do catálogo de estrelas K e M do *Kepler*.

De modo a tornar esta tese auto-contida, começamos por abordar conceitos essenciais em estatística, aprendizagem estatística, aprendizagem de máquinas, e astrofísica. Em seguida, descrevemos os materiais e a metodologia. Analisamos as variáveis em questão, investigando as relações entre as variáveis independentes e a resposta. Agrupamos as variáveis independentes de acordo com a sua natureza e métodos com os quais foram obtidas, e criamos onze conjuntos de dados a partir delas. Formalizamos então o problema, propondo abordagens baseadas em florestas aleatórias e *extreme gradient boosting* para criar modelos de regressão que nos possam ajudar a atingir os nossos objetivos de investigação. Descrevemos os conjuntos de experiências que realizámos utilizando as florestas aleatórias e o *gradient boosting*, e apresentamos os resultados obtidos com elas. A estratégia de avaliação baseia-se na comparação de métricas de *goodness of fit*, tais como o coeficiente de determinação ajustado, e o valor absoluto médio do erro relativo. Os nossos modelos demonstram um desempenho preditivo comparável a outros modelos semelhantes publicados recentemente, ao mesmo tempo que são fiáveis e computacionalmente "baratos". Além disso, a nossa abordagem apresenta a vantagem adicional de utilizar conjuntos de dados com menos preditores, e, principalmente, não utilizamos períodos de rotação obtidos com métodos clássicos como variáveis independentes. Indicamos os conjuntos das variáveis mais importantes para construir modelos sólidos de aprendizagem da máquina que possam ser usados para automatizar o processo

de previsão do período de rotação de estrelas de classe espetral K e M. Com base nos resultados obtidos com este estudo, concluímos que os melhores modelos obtidos com a metodologia proposta são competitivos quando comparados com as abordagens mais avançadas.

Palavras-chave: *gradient boosting*, aprendizagem de máquinas, floresta aleatória, regressão, parâmetros estelares, período de rotação estelar, estrutura estelar.

Contents

List of Tables

List of Figures

Acronyms

10CV 10-fold cross-validation

AB adaptive boosting
ACF autocorrelation function
AI artificial intelligence
ANN artificial neural network

CS composite spectrum
CV cross-validation

DL deep learning
DT decision tree

GB gradient boosting
GLM generalised linear model
GWPS global wavelet power spectrum

IG information gain

*k***NN** *k*-nearest neighbours

MAE mean absolute error
MARE mean absolute relative error
ML machine learning
MSE mean squared error

`NaN` not-a-number

PSD power spectral density

RF random forest

RMSE root mean squared error

SVC support vector classifier
SVM support vector machine

TS time series

WPS wavelet power spectrum

XGBoost extreme gradient boosting

Prediction of Stellar Rotation Periods Using Regression Analysis

CHAPTER 1

Introduction

Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information upon it.
— Samuel Johnson (1709-1784)

We can imagine that this complicated array of moving things which constitutes "the world" is something like a great chess game being played by the gods, and we are observers of the game. We do not know what the rules of the game are; all we are allowed to do is to watch the playing. Of course, if we watch long enough, we may eventually catch on to a few of the rules.
— Richard Feynamn (1918-1988)

Contents

TIME DOMAIN ASTRONOMY is the branch of Astronomy devoted to the study of variable phenomena in celestial bodies. It focuses on systems that measurably fluctuate during the observation period, with timescales typically ranging from fractions of a second to decades. Time domain astronomy addresses both predictable and random events (*e.g.*, Carroll and Ostlie, 2017). Of particular interest for this project is the rotation of stellar bodies, which produces periodic variations in their appearance, as is evident from the observation of the Sun's photosphere.

Stars have been one of the hottest topics since the dawn of Astronomy. With the rapid development of Astrophysics in the early 19^{th} century, after the works of Wollaston (Usselman, 2022) and von Fraunhofer (Encyclopaedia Britannica, 2022), they quickly became one of the main objects of celestial studies, as they act as freely available living laboratories, where many physical phenomena, otherwise hard or impossible to reproduce on Earth, take place (*e.g.*, Gomes, 2016).

Examples include nuclear fusion, strong and large-scale magnetic interactions, mass ejection, and planet formation.

Solar type stars, *i.e.*, low mass stars with convective outer layers, can exhibit magnetic activity (*e.g.*, Brun and Browning, 2017), which is manifested by the emergence of magnetic spots at their surfaces. Stellar spots can be detected by measuring the light curve of a star over time, since the brightness becomes modulated as spots come in and out of the visible hemisphere of the star. Consequently, such spot modulation is an important indicator of stellar magnetic activity and stellar rotation (*e.g.*, Strassmeier, 2009). The accuracy and precision at which stellar rotation is measured is crucial for the study of the evolution of stars and, ultimately, of the Galaxy, as well as to properly characterise planetary systems. The rotation period of a star is correlated with its age: it is known that solar type stars spin down during their main-sequence evolution, so that the period of rotation of young solar stars can be used to constrain stellar ages, through the *gyrochronology relations* (Skumanich, 1972; García et al., 2014). However, for stars older than the Sun, ages determined by the gyrochronology method do not agree with asteroseismic ages (Jennifer L Van Saders et al., 2016) and those inferred from velocity dispersion (Angus, Beane et al., 2020), making it necessary to improve the gyrochronology relations. The rotation period of a star is important to understand the transport of stellar angular momentum, a process which, on the one hand, is not yet sufficiently understood (Aerts, Mathis and Rogers, 2019), and, on the other hand, is essential to correctly estimate the age of the stars (Eggenberger et al., 2009). The latter is, in turn, vital to characterise planetary systems (Huber et al., 2016), the evolution of which is driven by tidal and magnetic effects between planets and their host stars (Benbakoura et al., 2019), and to understand how the Milky Way is evolving (Miglio et al., 2013).

The huge amount of astronomical photometric data released during the last three decades has recently motivated the use of machine learning (ML) techniques to handle and analyse them. The advent of new large sky surveys, and the need to process a large number of targets simultaneously is making the manual handling of astrophysical data impracticable and the use of artificial intelligence (AI) techniques increasingly popular (*e.g.*, Pichara Baksai, Protopapas and Leon, 2016; Biehl et al., 2018). A stellar light curve is no more than a *time series* of photometric data of a star, *i.e.*, a sequence of surface stellar fluxes collected at successive points in time. Over the last decade we have seen the emergence of numerous observations contemplating high-quality, long-term, and nearly continuous photometric stellar data. Examples are the *Kepler* space observatory (W. Borucki et al., 2009), with almost 200 000 targets (Mathur, Huber et al., 2017), the reborn *Kepler* K2 mission (Howell et al., 2014), with more than 300 000 targets (Huber et al., 2016), and the Transiting Exoplanet Survey Satellite (TESS, Ricker et al., 2015), which gathered light curves with time spans from 25 days to one year for tens of millions of stars. Such an amount of observations require automatic procedures to process and extract information from them, and machine learning methods can be used to fulfil that task.

In order to choose a machine learning (ML) technique, one has firstly to select the data from available sources and study them carefully. Two approaches can be followed: to use the photometric time series data directly as input, or previously convert the light curves into structured data that

is suitable to be represented by a set of variables or features in tabular form. Machine learning models can be trained from those two types of data (unstructured and structured) and automate processes that otherwise would be tedious or would require too much manpower.

The first case (unstructured data) is nowadays tackled with special types of *artificial neural networks* (Haykin, 2009), reinforcement algorithms that need little to no pre-processing of the data. Blancato et al. (2022) used a *deep learning* approach and applied *convolutional neural networks* to predict stellar properties from *Kepler* data. However, training neural networks typically requires heavy computational resources.

The second scenario (using structured data) is solved by resorting on algorithms that can use tabular data to carry out unsupervised (clustering) and supervised tasks (classification and regression). Breton et al. (2021) applied *random forests* (Breiman, 2001) on *Kepler* data in tabular form to create three ML classifiers, which allow to detect the presence of rotational modulations in the data, to flag close binary or classical pulsators candidates, and to provide rotation periods of K and M stars from the *Kepler* catalogue. The team used 159 inputs of different kinds to train the classifiers: rotation periods, stellar parameters such as mass, effective temperature, and surface gravity (just to name a few), and complementary variables obtained via wavelet analysis, the autocorrelation function of light curves, and the composite spectrum. They claim accuracies of 95.3 % when willing to accept errors within 10 % of the reference value, and of 99.5 % after visual inspection of 25.2 % of the observations. One particularity of their work is that they have used rotation periods as input variables to train their models. We expected those features to be highly correlated with the target variable (the final rotation periods), and that aroused our curiosity, on the one hand, to explore their data set and determine what is that level of correlation; and, on the other hand, to try to train ML models without those rotation period input variables and compare the performance of the models.

In this thesis, we concentrate on the prediction of rotation periods of K and M stars from the *Kepler* catalogue (W. Borucki et al., 2009; W. J. Borucki et al., 2010), resorting on ML approaches. The main challenges will be related with the selection of suitable data, the size of the data set, the choice of the best ML methods, the optimisation of their parameters, and the training of the models. We foresee that time and computational resources will be an important constraining factor in the development of this project, especially during the learning of the models.

1.1 Research Goals

In the previous section, we addressed the importance of time domain astronomy in the context of the study of stellar physics, including our Sun, of the Galaxy, and in the search for extrasolar planets. We described the main difficulties astronomers currently are confronted with in face of the large amount of data that needs to be explored. We also pointed out possible approaches to analyse the data, and the challenges that ML strategies used to predict stellar rotation periods entail. In this context, the main goal of this thesis is the following:

Main Goal

To determine the surface rotation period of a great number of K and M stars from a set of predictors in tabular form, resorting on supervised machine learning methods.

By predicting the rotation period for a large number of stars using machine learning and statistical analysis, we aim at improving the predictive performance of current models, namely the one built by Breton et al. (2021), and at making available an efficient and computational "cheap"* tool to automatically predict reliable stellar rotation periods from the *Kepler* catalogue, that can be applied to thousands of stars, possibly of other spectral classes than K and M.

The research goal can be decomposed into four questions:

Research Question 1

Which independent variables evince the highest level of correlation with the target variable?

Research Question 2

How does a regression ML model trained on input variables classically used to estimate stellar rotation periods compare to the classifier developed by Breton et al. (2021)?

Research Question 3

Which sets of input variables are mandatory for obtaining a reliable ML model, with good predictive performance?

Research Question 4

Is it possible to build an optimal subset of predictors from the set of available explanatory variables from which robust regression ML models, with good predictive performance, can be trained?

The first mandatory step before addressing these questions was the selection of the data with which models would be trained (section 3.1). We compared the possibilities of using real-world *vs*. synthetic data. We also defined the quality assessment strategy, so that the predictive performance of the models could be characterised (section 3.3.3).

In order to answer to the first question, we performed exploratory data analysis and calculated the correlations between the input variables and the target variable (section 3.1.2).

The second research question was addressed by building ML regressors from the data available, and by assessing their predictive performance using the quality metrics identified beforehand (sections 4.1 and 4.2).

The third question was answered by creating several subsets of predictors, grouped by their type and the processes from which they were obtained, by training similar regressors using each

* By "computational cheap" we mean that there is no need for supercomputers or network computing to get results in a timely manner.

of those subsets, and evaluating the predictive performance of the models thus obtained (same sections as for the previous research question).

Finally, the last question was answered by training the models obtained previously on the variables of greatest importance (section 4.3).

The code developed for the experiments described above will be made public, in support for reproducible science.

1.2 Thesis Outline

We close this introductory chapter, by outlining the organisation of the remaining of the manuscript.

Chapter 2, Background. The fundamental background is provided in chapter 2, which is divided into three sections. The first section pertains to concepts related to statistics and statistical learning, such as data, variables, models, correlation, common metrics of performance assessment, resampling techniques, and strategies for hyperparameter optimisation. The second section describes the most common supervised learning techniques, including linear and non-linear models, both in the classification and regression scenarios. The third section covers basic astronomical concepts, which act as an aid to contextualise the research problem and better understand the meaning of some variables used during our study.

Chapter 3, Materials and Methods. Chapter 3 is composed of three sections. The first one presents the materials used to execute the experiments, *i.e.*, the data, and explains the motivation for our choice of the type of data selected. We describe the steps we performed to explore and engineer the data, and the statistical analysis of the explanatory variables, with emphasis on the correlations between those and the target variable. In the second section, we formalise the scientific problem we want to address, and describe the two ML approaches we have used to address the main goal of the thesis and answer the research questions. In the third section, we detail the experimental design, describing which strategy we followed to optimise the hyperparameters of the models during the training phase, and present the performance assessment methodology.

Chapter 4, Prediction of Stellar Rotation Periods. In chapter 4 we present the results of our contributions to the application of supervised ML approaches to predict the rotation periods of stars. The chapter is divided into four sections. The first two sections pertain to the results obtained with random forests and gradient boosting using the first nine data sets described in chapter 3. The third section is similar to the two previous ones, but the analysis is done on the models built on top of the two extra data sets created after the experiments of the first two sections of this chapter. The fourth section is dedicated to discuss the results obtained before.

Chapter 5, Conclusions. The thesis concludes in chapter 5, where we summarise our contributions, we answer the research questions put forward in this chapter, point out open issues, and present future research directions.

CHAPTER 2

Background

All men have stars, but they are not the same things for different people. For some, who are travelers, the stars are guides. For others they are no more than little lights in the sky. For others, who are scholars, they are problems...
— Antoine De Saint-Exupéry (1900-1944)

Contents

IN THIS CHAPTER we describe the essential theoretical notions that support our contributions. Concepts underpinning others that are used in the dissertation, or that are relevant to understand the context, are also briefly explained. We start with basic definitions related to statistics and statistical learning, sailing around the concepts of data, variables, models, and correlation. We then define common metrics used to assess the performance of ML models, introducing a few developed within the framework of this thesis. After that, we describe tree-based regression learning techniques, that were used in this thesis to train several models from real data. We finish

the chapter by introducing some concepts related to time domain astronomy, that are important to understand the variables contained in the main data set and the motivation for the work carried out in the framework of this thesis.

2.1 Statistics and Statistical Learning

2.1.1 Basic Concepts

Data, variables, and models. Ultimately, this thesis is about learning models from astronomical data. We intend to predict a quantitative outcome measurement, the *response*, *target variable*, or *dependent variable*, which in our case is the rotation period of stars, from a set of *inputs*, *explanatory variables*, *features*, *predictors*, or *independent variables*, such as the stellar mass, its effective temperature, the photometric activity index, or the amplitude of the first Gaussian fitted in the composite spectrum.

We start with a set of structured, tabular data, containing measurements for the features and for the response variable, for several *observations*, *instances*, *cases*, or *objects*, which, in this context, correspond to thousands of stars collected by the *Kepler* space observatory. Computationally speaking, these tabular data are organised into *data frames*, where the columns correspond to the variables (features plus the response), and the rows to the observations. This data set is split into the *training* and *testing* sets. The former is used to build a prediction *model*, which in turn will be applied to predict the rotation period of unseen stars, provided they are input as a set of structured data similar to the training set, *i.e.*, in a tabular form, containing at least some of its features (the more the better). The testing set acts as the never-before-seen objects, with the advantage of containing the outcome (that is not given to the model), which can be used to assess the predictive performance and quality of the model, by comparing the predicted values with the true outcomes. A good model is one that accurately predicts the response.

What we just described is known as *supervised learning*, where the target variable is used to guide the learning process, as opposed to *unsupervised learning*, in which no measurements of the response variable are used—in this case, the values of the features are used to calculate their relevance in the model (Torgo, 2011).*

Variables can be of different nature. They are *continuous* or *quantitative*, when they can be represented by real numbers, *i.e.*, when some values are bigger than others and close measurements are close in nature (James et al., 2013); they are *categorical*, *qualitative*, or *discrete*, when they are non-numerical or assume values in a finite set, *i.e.*, they take on values in one of k different classes or categories—in this case, they are commonly referred to as *factors*; they can also be

* There are two more major categories of learning: *semi-supervised*, and *reinforcement* learning. The former is similar to supervised learning, but both labelled and unlabelled data are used during the training of the model; the latter focuses on processes where a learner can observe the surrounding environment, perform actions, and get rewards or penalties in return, so that it can learn the best strategy in order to get the most reward over time. Examples of application are text and photo classifiers for semi-supervised learning, and artificial neural networks and self-driving car algorithms for reinforcement learning. We refer the reader to, for example, Hastie, Tibshirani and J. Friedman (2009), James et al. (2013), Kuhn, Johnson et al. (2013) and Géron (2017) for a comprehensive explanation about these types of learning.

ordered categorical, when there is an order within the values, but no underlying metric between them (Hastie, Tibshirani and J. Friedman, 2009).

A prediction task is named according to the nature of the response. When the latter is quantitative, the task is known as a *regression* problem; when the target variable is qualitative, it is known as a *classification* task (if instances are classified into one of two classes), or *multiclass classification* or *multinomial classification* if the cases are classified into three or more classes. The problem addressed in this thesis is of regression type.

Specifying mathematically, without loss of generality, for a regression problem, we are assuming there is a relationship between a quantitative response y and a set of p predictors $\mathbf{x} = (x_1, x_2, \ldots, x_p)$, such that

$$y = f(\mathbf{x}) + e, \tag{2.1}$$

where f is an unknown fixed function of $(x_1, x_2, \ldots, x_p)$, and e represents a random *error term* independent of $\mathbf{x}$, with zero mean. The function f can be estimated, but never known completely. When the inputs $\mathbf{x}$ are readily available, in the presence of an estimation for f, $\hat{f}$, the output can be predicted by

$$\hat{y} = \hat{f}(\mathbf{x}). \tag{2.2}$$

Typically, we are not concerned with the exact form of $\hat{f}$, provided it yields accurate predictions for y.* This is the situation we have at hands in the framework of this project: we have measurements for a set of features and for the response, which we will try to predict by learning a model using two different approaches (to be explained in section 2.2.7). We will be mainly concerned with the accuracy of the models obtained, not so much in understanding the exact shape of f.

Since $\hat{f}$ is not a perfect estimate of f, the inaccuracy of the estimation will give rise to two types of error, or *noise*, upon which the accuracy of $\hat{y}$ as a prediction of y depends: *reducible* and *irreducible*. While the former can be mitigated by using an appropriate ML technique to estimate f, the second cannot be diminished because it is impossible to reduce the error introduced by e. In fact, since y is also a function of e, and the latter cannot be predicted by $\mathbf{x}$ (by definition), even if we could estimate perfectly f, $\hat{y} = f(\mathbf{x})$, the prediction would still contain some error and, so, the accuracy of the predictions is affected by the variability associated with e, as well. The error term may be originated from unmeasured variables that are important to predict y, or from unmeasurable variation (Hastie, Tibshirani and J. Friedman, 2009).

When estimating f from the training set, *i.e.*, when training the model, we usually adopt a *parametric* approach, in which the problem is reduced down to estimating a set of parameters. The optimal set of values for the parameters will avoid both *underfitting* and *overfitting*. A model is said to underfit the data when it is too rigid, and is not able to capture the relationship between the features and the response accurately; it is said to overfit the data when it follows the errors too closely, thus performing poorly on unseen data.

* In some domains of application, such as in the health area, the transparency and interpretability of a model is very important, and $\hat{f}$ should not be treated simply as a *black box*.

Association and correlation. We say that an *association* exists between two variables when values of one of the features are more likely to occur with particular values of the other.

Correlation is a statistical quantity that measures the level of relationship, causal or not, between two random variables. In statistics, correlation is a summary measure describing the strength of linear association, that is, the extent to which two variables are linearly related. However, in its broadest definition, it may indicate any type of association between any two variables. It is usually expressed by means of the correlation coefficient, r , which measures the strength and direction of a linear relationship. It varies between -1 and $+1$: these limiting values indicate a perfect negative and positive linear relation, respectively, while 0 indicates no linear relationship between two different variables (Agresti, Franklin and Klingenberg, 2018).

Generally, it is a good practice to avoid data with highly correlated variables. Redundant features frequently add more complexity than information to the model, decreasing its interpretability, and making its training process more costly. In some models, such as linear regression and neural networks, the presence of highly correlated variables can lead to collinearity issues, such as instabilities and numerical errors, which can greatly increase the response variance and generally degrade the performance of the model (Kuhn, Johnson et al., 2013). A correlation analysis of the data will be presented in section 3.1.2.

2.1.2 Performance assessment

The performance of a ML model on a given data set is assessed by quantifying to which extent the predictions are close to the true values of the response for the set of observations (James et al., 2013). Given the regression nature of the problem in this project, we used six metrics to assess the predictive quality and the goodness of fit* of the learners: (*a*) the *root mean squared error* (RMSE) and the *mean absolute error* (MAE); (*b*) the mean of the absolute values of the residuals, μ_{err}; (*c*) two interval-based "accuracies" developed within the framework of this thesis; and (*d*) the adjusted coefficient of determination, R^2_{adj}. We are going to briefly describe each of them in the following.

Mean squared error. In a regression setting, the most commonly-used metric to assess the performance of a model is the *mean squared error* (MSE). The MSE of a point estimator $\hat{\theta}$ is the expected squared deviation of the estimator from the true or real value θ of the parameter it estimates:

$$\text{MSE}(\hat{\theta}) = \text{E}\left[(\hat{\theta} - \theta)^2\right]. \tag{2.3}$$

The deviations of the estimator from the true values, the *errors*, are commonly referred to as the *residuals*. With the notation used in section 2.1.1, eq. (2.3) becomes:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left[(y_i - \hat{y}_i)^2\right], \tag{2.4}$$

* The *goodness of fit* of a model is a metric giving an indication of how well the model fits a set of observations (Nolan and Speed, 2001).

where $\hat{y}_i = \hat{f}(\mathbf{x}_i)$ is the prediction for the i^{th} observation given by $\hat{f}$, and n is the number of cases present in the data set.

The MSE is a measure of the distance between $\hat{\theta}$ and θ ($\hat{f}(\mathbf{x}_i)$ and y, respectively). It is usually interpreted as how far, on average, the residuals are from zero, that is, its value represents the average distance between the model predictions and the true values of the response (Kuhn, Johnson et al., 2013). In general, a smaller MSE is indicative of a better estimator (Ramachandran and Tsokos, 2020): the MSE will be small if the predictions and the responses differ a little, and will be typically large if for some instances they are not close to each other. The MSE can be computed on both the training and testing data, but we will be mostly interested to measure the performance of the model on unseen data (the testing set)—the one with the smallest testing MSE will have the best performance. When a learner yields a small training MSE but a large testing MSE, that is an indication that it is overfitting the data and, hence, that a less flexible model would have yielded a smaller testing MSE (James et al., 2013).

The MSE can be decomposed into two quantities:

$$\text{MSE}(\hat{\theta}) = \text{Var}(\hat{\theta}) + \left[\text{B}(\hat{\theta})\right]^2, \tag{2.5}$$

where $\text{Var}(\hat{\theta})$ is the *variance*, $\text{B}(\hat{\theta}) = \text{E}(\hat{\theta}) - \theta$ is the *bias*, and $\text{E}(\hat{\theta})$ is the *expected value* or *expectation* of the point estimator $\hat{\theta}$ (Ramachandran and Tsokos, 2020). The bias of an estimator $\hat{\theta}$ tells us how far $\hat{\theta}$ is on average from the real value θ. The squared of the bias term in eq. (2.5) corresponds to the reducible part of the error term of eq. (2.1), while the variance corresponds to the irreducible error.

A common measure of the differences between estimations and real values is the RMSE, which is no more than the square root of the MSE:

$$\text{RMSE}(\hat{\theta}) = \sqrt{\text{MSE}(\hat{\theta})}. \tag{2.6}$$

The RMSE has the advantage over MSE of having the same units as the estimator. For an unbiased estimator, it corresponds to the square root of the variance, or *standard deviation*.

Similarly to the MSE, in general, smaller values of the RMSE indicate a better estimator, but because this metric is dependent on the scale of the variables used, comparisons are only valid across models created with the same data set (Hyndman and Koehler, 2006).

The bias-variance trade-off. Assuming that the data points are statistically independent, and that the residuals have a theoretical distribution with zero mean and constant variance σ^2, given a previously unseen test observation $(\mathbf{x}_0, y_0)$, not used to train the ML method, the *expected testing MSE* can always be decomposed as the sum of three fundamental terms: the variance of $\hat{y}_0 = \hat{f}(\mathbf{x}_0)$, the squared bias of $\hat{y}_0$, and the variance of the error term (e), σ^2. Mathematically,

$$\text{E}\left[(y_0 - \hat{y}_0)^2\right] = \text{Var}(\hat{y}_0) + [\text{B}(\hat{y}_0)]^2 + \sigma^2, \tag{2.7}$$

where $\mathrm{E}\left[(y_0 - \hat{y}_0)^2\right]$ corresponds to the expectation of the testing MSE that we would get if f was repeatedly estimated on a large number of training sets, and each was tested on x_0 (James et al., 2013; Kuhn, Johnson et al., 2013). An overall value of this quantity can be obtained by averaging it out over all $\mathbf{x}_0$ values of the testing set.

The variance term refers to the variability of $\hat{f}$ when estimated using different training sets. The bias term is related to the error resulting from approximating a real-life problem by a simpler model, *i.e.*, it reflects how close the functional form of the model can get to the true relationship between the features and the response. The challenge is to find a model for which the variance and the square of the bias are simultaneously low. While the expected testing error can be minimised this way, it can never go below the irreducible error which, in eq. (2.7), is translated by $\sigma^2 = \mathrm{Var}(e)$. As a general rule, more flexible learners tend to be characterised by increased variance and smaller bias than more rigid models. Equation (2.7) is known as the *bias-variance trade-off*.

Mean absolute error. The *mean absolute error* (MAE) of an estimator $\hat{\theta}$ is the average of the absolute values of the errors:

$$\mathrm{MAE}(\hat{\theta}) = \mathrm{E}\left[|\hat{\theta} - \theta|\right]. \tag{2.8}$$

Using the notation of section 2.1.1, this equation becomes:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{2.9}$$

where all quantities have the same meaning as previously. The MAE is a measure of the errors between the predicted and the observed values, and it uses the same scale as the estimates and the true values. Therefore, it cannot be used to make comparisons between models created on top of different data sets.

The MAE has the advantage over the RMSE of being influenced by the error in a direct proportion to its absolute value (Pontius, Thontteh and H. Chen, 2008).

Mean absolute relative error. The *mean absolute relative error* (MARE) is given by

$$\mathrm{MARE} = \mu_{\mathrm{err}} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|, \tag{2.10}$$

where n is the number of observations, $y_i = f(x_i)$ and $\hat{y}_i = \hat{f}(x_i)$. It is similar to the MAE, but the mean absolute value is computed on the relative residuals rather than the magnitude of the residuals. The MARE is a measure of the mean relative error of the model, giving a percentage estimate of how wrong the model is, on average.

Interval accuracies. The problem we tackled in this thesis focus on a regression setting. One of the goals of the project is to compare the performance of our models with the one developed by Breton et al. (2021). However, the latter is a classifier and, therefore, we felt the need to develop a metric that would allow us to compare the results. We devise an interval-based error function

and an "accuracy" metric to bridge the gap between regression and classification settings. This accuracy can take two forms, which are equivalent to each other, as we will demonstrate in what follows.

The first form of the "accuracy" metric is inspired on the standard *error rate*, but it is calculated using intervals around the reference values of the response variable:

$$\text{acc}_x = \frac{1}{n}\sum_{i=1}^{n} I\left(\hat{y}_i \in \left[y_i - \frac{x}{2}, y_i + \frac{x}{2}\right]\right), \tag{2.11}$$

where, as before, $\hat{y}_i = \hat{f}(x_i)$, $y_i = f(x_i)$, and n is the number of observations. Here, x is a fraction width, and $I(\hat{y}_i \in \text{interval})$ is an *indicator function*, that equals 1 if $\hat{y}_i \in \left[y_i - \frac{x}{2}, y_i + \frac{x}{2}\right]$, and zero otherwise. When $I(\hat{y}_i \in \textit{interval}) = 1$, than a correct outcome was predicted for the i^{th} observation, within $x \cdot 100\,\%$ of the reference value, and it amounts saying that the classification was correct (in a classification scenario); otherwise, the outcome was incorrectly predicted within an interval of $x \cdot 100\,\%$ width of the reference value, and it will count as a misclassification.

By using the *interval accuracy* metric of eq. (2.11), we are able to convert the results of a regression problem into a classification setting: we get an *event*, *occurrence*, or *match* every time a predicted value lies within the interval centred on the reference response value, and a *non-event*, *non-occurrence*, or *no-match* otherwise. This way, we are able to estimate the accuracy of a model when we are willing to accept an error of $x \cdot 100\,\%$.

The other form of the *interval accuracy* is underpinned by an *interval-based version of the residuals*, ε_x. The latter estimates the prediction error in a regression task when we are willing to accept an error of $x \cdot 100\,\%$, and it is given by:

$$\varepsilon_x = \begin{cases} 0, & |y_i - \hat{y}_i| \leq x \cdot y_i \\ y_i - \hat{y}_i, & \text{otherwise,} \end{cases} \tag{2.12}$$

where $y_i = f(x_i)$ and $\hat{y}_i = \hat{f}(x_i)$ are respectively the reference and the predicted values, and x is the fraction width of the *zeroing interval*. When x is zero, the metric returns the simple residuals. Figure 2.1 illustrates the interval-based error function of eq. (2.12) computed on simulated data, obtained from an uniform distribution of 100 values varying between 0 and 45. Similarly to eq. (2.10), this error function, when normalised by the reference values, can be used to approximately estimate how much a model is wrong, on average, for a given *zeroing-interval* width.

An *error-interval* "accuracy", $\text{acc}_x^{\varepsilon}$, can be devised using the interval-based residuals of eq. (2.12) as inspiration:

$$\text{acc}_x^{\varepsilon} = \frac{1}{n}\sum_{i=1}^{n} I\left(\varepsilon_x = 0\right), \tag{2.13}$$

where x and n are the same as before. In this case, the indicator function equals 1 if the interval-based error is zero, *i.e.*, whenever $\varepsilon_x = 0$, and zero otherwise; that is, we consider an *event* every time the error is equal to zero in eq. (2.12), and a *no-event* otherwise.

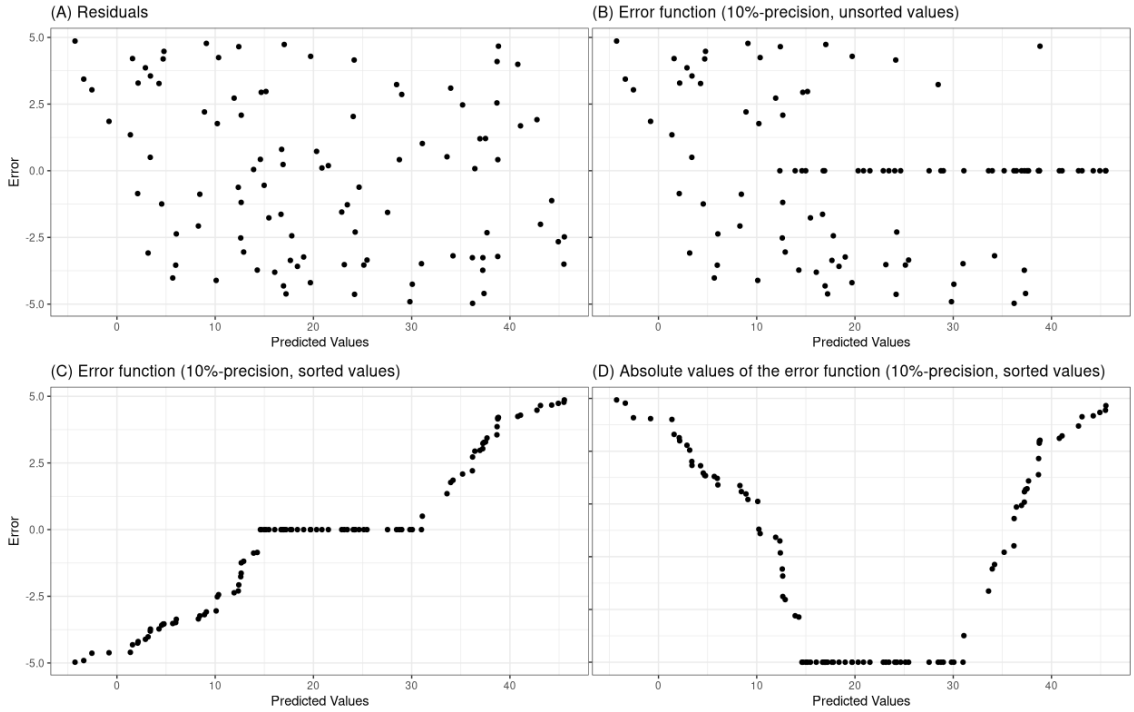

Figure 2.1: Graphical representation of the interval-based error function computed on simulated data. Panel (A) plots the residuals; panels (B), (C), and (D) similarly illustrate the 10 %-width error function, respectively on the unsorted data, sorted data, and absolute values of the sorted data. In all panels, the x-axis represents the predicted values, while the y-axis indicates the error.

As expected, the two aforementioned forms of the interval-based accuracy are equivalent, except for a scale factor. In fact, considering acc_x, we have an event every time $\hat{y}$ verifies the inequality

$$y - \frac{x}{2} \cdot y < \hat{y} < y + \frac{x}{2} \cdot y, \tag{2.14}$$

which is equivalent to

$$\left(1 - \frac{x}{2}\right) \cdot y < \hat{y} < \left(1 + \frac{x}{2}\right) \cdot y. \tag{2.15}$$

On the other hand, considering $\text{acc}^{\varepsilon}_{x/2}$, an event occurs whenever

$$|y - \hat{y}| < \frac{x}{2} \cdot y, \tag{2.16}$$

which equates to eq. (2.15). Hence, we conclude that $\forall x \in \mathbb{R}^+_0$,

$$\text{acc}_x = \text{acc}^{\varepsilon}_{x/2}. \tag{2.17}$$

For example, $\text{acc}_{0.10} = \text{acc}^{\varepsilon}_{0.05}$, that is, the interval accuracy measured on a 10 %-width interval, as defined by eq. (2.11), is exactly the same as the 5 %-error interval accuracy, as given by eq. (2.13).

This metric has the disadvantage of penalising too much a model for which the predicted values fall outside but close to the limits of the reference intervals. In those cases, a model will

report a low accuracy, while having most of its predictions falling very closely to the accuracy intervals. However, this limitation can be mitigated by investigating how much the accuracy increases when the widths of the intervals are increased. We will be using this metric when assessing the performance of models in chapter 4.

Coefficients of determination. Another common method for characterising a model's predictive capabilities is the *coefficient of determination*, R^2. The coefficient of determination is a measure of the goodness of fit of a model, *i.e.*, it is a statistic used to quantify how well a model describes the data. It indicates the fraction of the variability of the target variable that has been accounted for by the predictors. That is, it is the proportion of the information in the data that is explained by the learner (Kuhn, Johnson et al., 2013). Therefore, it can also be related to the fraction of unexplained variance (not captured by the model) of the response variable. In regression, it is an indication of how well the model predictions approximate the real data points (Casella and R. L. Berger, 2001). It is defined by (Kvålseth, 1985):

$$R^2 = \frac{\text{RegSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}, \tag{2.18}$$

where RegSS is the *regression sum of squares*, RSS is the *residuals sum of squares*, and TSS is the *Total Sum of Squares*.*

The coefficient of determination usually ranges between 0 and 1. A perfect fit of the data—that is, all variability in the target is explained by the fitted model—is indicated by an R^2 of 1, while the absence of "linear" relationship is indicated by an R^2 of 0. Nevertheless, there are situations, such as when the predictions that are being compared to the true outcomes have not been estimated with a model build upon those data, when R^2 can yield negative values. An example is when R^2 is computed using testing data, which were not used to build the model. Models with a negative R^2 have worse predictions than the *baseline model*, which always predicts the average of the outcome.

The coefficient of determination is not a fair criterion to compare learners with a different number of explanatory variables, because it never decreases when new predictors are added to a model (it can only remain the same or increase). A fair measure of the goodness of fit of a model is the *adjusted coefficient of determination*:

$$R^2_{\text{adj}} = 1 - \frac{(n-1)}{n-p-1} \cdot (1 - R^2), \tag{2.20}$$

* The total sum of squares (TTS) can be decomposed into the sum of the variation due to regression (RegSS) and the residual variation (RSS):

$$\begin{gathered} \text{TSS} = \text{RegSS} + \text{RSS} \\ \sum_{i=1}^{n} (y_i - \bar{y})^2 = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 , \end{gathered} \tag{2.19}$$

where $\bar{y}$ indicates the mean of the response. The TSS represents the response total variability, *i.e.*, the total variation of y; the RegSS and the RSS represent the response variability that is and that is not explained by the model, respectively (Casella and R. L. Berger, 2001).

where R^2 is the coefficient of determination, defined in eq. (2.18), n is the number of observations, and p is the number of predictor variables (Baron, 2019).

Resampling Techniques

When used to estimate model performance, resampling techniques all have in common the fact that a subset of observations are used to fit a model, and the remaining instances are used to assess the quality of the model. This process is repeated several times, and the results are aggregated and summarised. The major differences between resampling techniques reside on the methods used to choose the subsamples (Kuhn, Johnson et al., 2013). In the following, we will consider two of those techniques, which were directly or indirectly heavily used during the work of this thesis.

k-fold cross-validation. In the *k-fold* cross-validation (CV), the training instances are randomly partitioned into k non-overlapping sets or folds of approximately equal size. One of the folds is held out as a validation set, and the remaining folds are combined in a training set to which a model is fit. After the performance of the model is assessed on the validation fold, the latter is returned to the training set, and the process is repeated, with the second fold being held out as the new validation set, and so on and so forth. The k performance estimates are summarised with measures of location and dispersion (usually the mean and the standard error), and they are commonly used to tune the model's parameters (Kuhn, Johnson et al., 2013; Hastie, Tibshirani and J. Friedman, 2009). Typically, the testing error is estimated by averaging out the k resulting MSE estimates (or any other quality metric suitable for the problem at hands).

The bias of the technique, *i.e.*, the difference between the predictions and the true values in the validation set, decreases with k. Typical choices for k are 5 and 10, but there is no canonical rule. Figure 2.2 illustrates a CV process with $k = 3$.

Bootstrap. Bootstrapping is a sampling technique in which data are randomly taken with replacement (Efron and Tibshirani, 1986). The sample has the same size as the data set from which it is taken, so some observations will not be included in the bootstrap sample (referred as the *out-of-bag* cases), while others will be included multiple times. Typically, in an iterative bootstrap resampling process, a model is built on the selected observations and is used to predict the out-of-bag cases.

Hyperparameter Tuning

A *hyperparameter* or *tuning parameter* is a parameter which cannot be directly estimated from the data, and that is used to tune and improve the performance of a ML model (Kuhn, Johnson et al., 2013). That is, it is a parameter of a learning algorithm, not that of a model.

Hyperparameters are mostly used to control how much of the data are fit by the model, so that, in an ideal scenario, the real structure of the data is captured by the model, but not the noise. They are set beforehand and remain constant during the training process. Hyperparameters are not

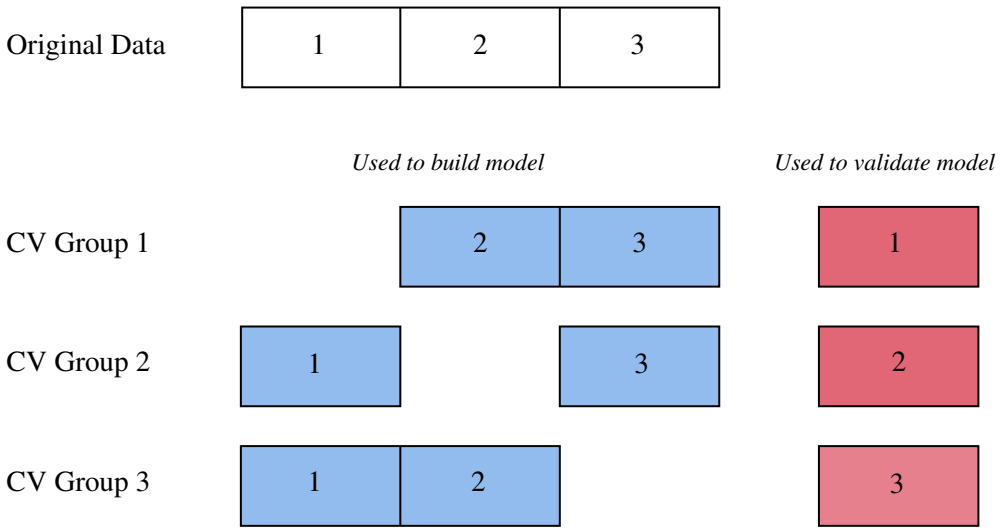

Figure 2.2: A schematic of a 3-fold CV process. The original training data is allocated to three non-overlapping subsets. Each fold is held out in turn as models are fit on the remaining training folds. Performance estimates, such as MSE or R^2 are estimated using the held-out fold in a given iteration. The CV estimate of the model performance is given by the average of the three performance estimates.

affected by the learning algorithm, but they have an impact on the speed of the training process (Géron, 2017; Raschka and Mirjalili, 2017).

There is no formula to calculate the optimal value nor unique rule to tune the parameters used to estimate a given model. The optimal configuration depends on the data set, and the best way to build a model is by testing different sets of hyperparameter values by resorting on resampling techniques, such as the ones introduced in section 2.1.2 (*Resampling Techniques*), page 18.

2.2 Common Supervised Learning Techniques

Supervised (machine) learning uses labelled data—a training set containing inputs and correct outputs—to train algorithms to perform predictive tasks, *i.e.*, to teach models to yield the desired output, such as to classify a set of observations into specific categories (classification problem), or to predict the value of a continuous target variable (regression problem). Recalling eqs. (2.1) and (2.2), the goal is to obtain a model that relates the response, y, to a set of p independent explanatory variables, $x_1, x_2, \ldots, x_p$, by approximating an unknown function f describing the relationship between them (Torgo, 2011). In this section, we will describe some of the most common supervised ML approaches, that can be applied in regression or classification problems, or in both of them. Linear and logistic regression, naïve Bayes, k-nearest neighbours, support vector machines, decision trees, tree-based ensembles (bagging, random forests, and boosting), and

artificial neural networks are examples of supervised learning algorithms, which will be described in more detail in the following.*

2.2.1 Linear Regression

In linear regression analysis, the goal is to model the relationship between a continuous random variable y (the response or target variable) and a set of p explanatory (observed, not random) variables $x_1, x_2, \ldots, x_p$ of any type, with the purpose of (a) evaluate the effect of the independent variables on the response, and (b) to forecast, *i.e.*, predict the values of y from the known values of $x_1, x_2, \ldots, x_p$. An *ordinary linear regression model* can be mathematically written as

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_p x_p + e, \tag{2.21}$$

where $b = (b_0, b_1, \ldots, b_p)$ are the unknown *coefficients* or *parameters*, and e is the random error that cannot be explained by the model. The parameters of the model are estimated during the training process, by minimising a function of the sum of the squared errors. The parameter b_0 is known as the *estimated intercept*.

When a model can be represented by eq. (2.21), it is said to be *linear in the parameters*. Other examples of this type of learners are *ridge regression*, *lasso*, and *elastic net*, which are penalised models, in the sense that one or two *regularisation terms* are included in eq. (2.21) in order to enhance the prediction accuracy and interpretability of the model. Ordinary linear regression estimates parameters that lead to a minimum bias, whereas lasso, ridge regression, and elastic net find estimates that yield the lowest variance possible (Kuhn, Johnson et al., 2013). Advantages of models following the form of eq. (2.21) are:

- They are highly interpretable: the estimated coefficient of a predictor being β means that a one unit increase in that predictor's value will, on average, increase the response by β units.
- The estimated parameters can provide further insights about relationships between the predictors.
- The statistical significance of the predictors can be assessed by computing the standard errors for the parameters, provided certain assumptions are made about the distribution of the model's residuals.

These models can only be applied in regression problems. Moreover, they are appropriate only when the relationship between the predictors and the response falls on a hyperplane. When it is not the case, that is, when the relationship is non-linear, these models might not capture it properly (Kuhn, Johnson et al., 2013).

* This is not an exhaustive list, and several more ML algorithms could have been included in it. We will be particularly interested in two tree-based ensemble methods (random forests and gradient boosting), which will be addressed in detail in chapter 3.

2.2.2 Logistic Regression

Logistic regression is a probabilistic model that aims at explaining or predicting a binary response variable y from a set of predictors $\mathbf{x} = (x_1, \ldots, x_p)$ of any type, given a set of observations (Hosmer and Lemeshow, 2013). That is, logistic regression models the probability that the target variable belongs to a particular category and, therefore, it is applied to classification problems.

If $q(\mathbf{x}) = \Pr\{y = 1 \mid \mathbf{x}\}$ is the probability of an event or of a specific class given the set of predictors $\mathbf{x}$, then we can model q by the *logistic function*:

$$q(\mathbf{x}) = \frac{e^{b_0+b_1x_1+\ldots+b_px_p}}{1+e^{b_0+b_1x+\ldots+b_px_p}}. \tag{2.22}$$

Being sigmoidal of the model terms, this equation varies between 0 and 1, and it can be rewritten as

$$\frac{q(\mathbf{x})}{1-q(\mathbf{x})} = e^{b_0+b_1x+\ldots+b_px_p}. \tag{2.23}$$

The left hand side of eq. (2.23) is known as the *odds* of the event (or the odds of belonging to the class). Logistic regression models the *log-odds* or *logit* of the event or of the class as a linear function (usually referred as the *linear predictor*):

$$\log\left(\frac{q}{1-q}\right) = b_0 + b_1x_1 + \ldots + b_px_p, \tag{2.24}$$

where the explicit dependence of q on $\mathbf{x}$ has been omitted to lighten the notation, and p, as before, is the number of predictors.

Logistic and ordinary linear regression models belong to a large class of methods called *generalised linear models* (GLMs), which includes several different probability distributions (Dobson and Barnett, 2018).

2.2.3 Naïve Bayes

The *Naïve Bayes* model is a simple probabilistic classifier based on Bayes' theorem with strong (naïve) independence assumptions between the predictors. For any given two events, A and B, Bayes' theorem states that

$$\Pr\{A \mid B\} = \frac{\Pr\{B \mid A\} \cdot \Pr\{A\}}{\Pr\{B\}}. \tag{2.25}$$

The Naïve Bayes classifier applies this theorem to calculate the probability that the response is class c_j given the predictors that have been observed:

$$\Pr\{y = c_j \mid \mathbf{x}\} = \frac{\Pr\{\mathbf{x} \mid y = c_j\} \cdot \Pr\{c_j\}}{\Pr\{\mathbf{x}\}}, \tag{2.26}$$

where $\mathbf{x} = (x_1, \ldots, x_p)$ is the set of p values of the predictors. $\Pr\{y = c_j \mid \mathbf{x}\}$ is called the *posterior probability* of the class. The components of eq. (2.26) are the following:

- $\Pr\{y = c_j\} = \Pr\{c_j\}$ — the *prior* expectation of the class c_j; it is the expected probability of the class based on what is known about the problem.
- $\Pr\{\mathbf{x}\}$ — the unconditional probability of the predictor values; it is formally calculated using a multivariate probability distribution.
- $\Pr\{\mathbf{x} \mid y = c_j\} = \Pr\{\mathbf{x} \mid c_j\}$ — the *conditional probability*; it is the likelihood of the testing observation given the class c_j, *i.e.*, it corresponds to the probability of observing the predictor values for the data associated with the class c_j.

The naïve Bayes model reduces greatly the complexity of the calculation of the probabilities of the feature values by assuming independence between the predictors, *i.e.*, by naïvely assuming that the attributes are class-conditional independent. Under this severe assumption, the unconditional probability of the observed evidence is

$$\Pr\{\mathbf{x}\} = \prod_{i=1}^{p} \Pr\{x_i\}, \tag{2.27}$$

which is a constant over all classes. Consequently, the most probable class, that is, the class $c_{\max}$ that maximises eq. (2.26), known as the *maximum posterior hypothesis*, depends only on the numerator of eq. (2.26). The conditional probability is calculated in a similar way:

$$\Pr\{\mathbf{x} \mid c_j\} = \prod_{i=1}^{p} \Pr\{x_i \mid c_j\}. \tag{2.28}$$

For categorical predictors, the individual probabilities are estimated using relative frequencies calculated from the training set; in the case of continuous features, it is assumed they follow a Normal distribution, whose mean and standard deviation are sampled from the training set.

In spite of these unrealistic and strong assumptions, a naïve Bayes model can be quickly trained, even for large training sets, and it can perform competitively in many scenarios (Kuhn, Johnson et al., 2013).

2.2.4 k-Nearest Neighbours

The *k-nearest neighbours* (kNN) algorithm is a *lazy learner*[*] based on distances between observations, that can be used in both classification and regression problems (Torgo, 2011). It attempts to estimate the conditional distribution of the response for a particular set of predictors before performing a prediction. Given a positive integer k and a prediction point x_0 (a test observation), kNN first identifies the set $\mathcal{N}_0$ of the k training observations that are closest to x_0. Then, in a classification problem, the conditional probability for a class j is taken as equal to the fraction of points in $\mathcal{N}_0$ for which the response is j:

$$\Pr\{y = j \mid x = x_0\} = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j), \tag{2.29}$$

[*] Lazy learners, instead of building a model from the training set, they simply store the data. Their main work happens when they perform predictions (Torgo, 2011).

where, as in section 2.1.2, I is the indicator function. The test point x_0 is classified to the class with the largest probability, using the Bayes rule. In a regression task, the approach is similar: kNN estimates $f(x_0)$ using the average of all the training responses in $\mathcal{N}_0$:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{x_i \in \mathcal{N}_0} y_i. \tag{2.30}$$

The prediction on x_0 can also be calculated using other summary statistics, such as the median.

The distances between the prediction point and the training observations are generally obtained from the Minkowski distance:

$$d = \left(\sum_{j=1}^{p} |x_{a_j} - x_{b_j}| \right)^{\frac{1}{q}}, \tag{2.31}$$

where p is the number of predictors, x_{a_j} and x_{b_j} are two individual samples, and $q > 0$. When $q = 2$, Minkowski's reduces to the Euclidean distance (Kuhn, Johnson et al., 2013; Liu, 2011).

2.2.5 Support Vector Machines

Support vector machines (SVMs) are one of the most successful and effective ML tools available, capable of performing linear and non-linear classification, regression, and outlier detection (Torgo, 2011; Kuhn, Johnson et al., 2013; Géron, 2017). These models produce non-linear boundaries by building a linear boundary in a large, transformed version of the predictor space (Hastie, Tibshirani and J. Friedman, 2009). While defining the boundary, the model allows for misclassifications, so that overfitting is avoided at the cost of increasing the bias (bias-variance trade-off). This results in overall improved predictions on new data.

The fundamental ideas behind these techniques are best understood using pictures and by considering the simple classification case of two classes. When classes can clearly be separated with a straight line, they are said to be *linearly separable*. Figure 2.3 illustrates an example of a data set with two linear separable classes. The blue solid line represents the decision boundary. On the left panel, this line clearly separates both classes and remains as far away from the closest training observations as possible. The *support vector classifier* (SVC) fits the widest (maximal) possible margin between the classes (represented by the shaded strip), which in this case has width $2M$ and is defined by the two parallel dotted lines. All instances lie outside and on the right side of the margin, and so the latter is called *hard margin*. The margin is fully determined (or "supported") by the instances located on its edge. These instances—three, in this case (two green and one red), circled in blue—are called the *support vectors*. They define both the margin and the decision boundary. Adding training instances outside the margin does not affect the model's predictions.

The SVMs are sensitive to the scale of the predictors, and so the maximal margin is most easily obtained by normalising the variables. However, sometimes it is hard or even impossible to find a hard margin. In this cases, more flexible models are used, allowing misclassifications in order to find a good balance between maximising the margin and limiting its violations. This is called *soft margin classification*, which is illustrated on the right panel of fig. 2.3. In this case, the points labelled as ξ_i^* are on the wrong side of the margin, while points on the correct side have $\xi_i^* = 0$.

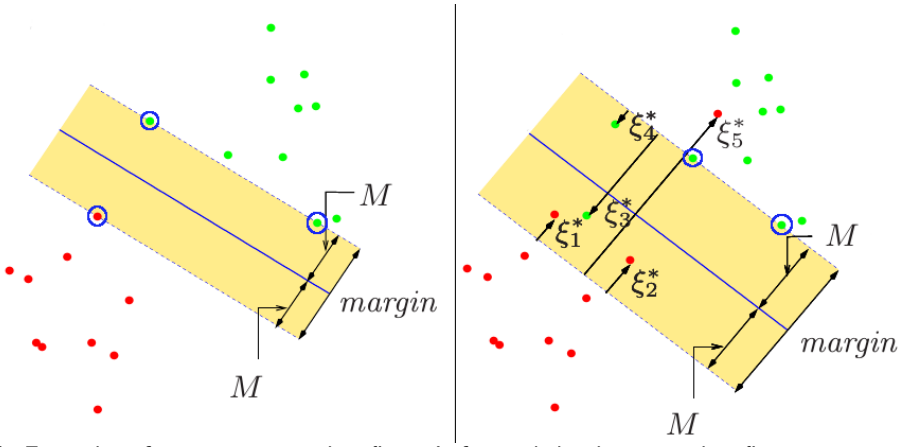

Figure 2.3: Examples of support vector classifiers. *Left panel*: hard margin classification, an example of a linear separable case. *Right panel*: soft margin classification, an example of a non-separable (overlap) case. The decision boundary is indicated by the blue solid line. The maximal margin, of width $2M$, is bounded by the two parallel dotted lines. The points labelled ξ_i^* are on the wrong side of the margin. Adapted from Hastie, Tibshirani and J. Friedman (2009).

The margin is maximised by keeping a total budget $\sigma_i \xi_i$ below a certain number. The choice of support vectors and number of misclassifications is done using CV.

When the data set is not linearly separable, a possible approach is to add more predictors, such as polynomials. However, that can increase too much the number of features to a point that the model becomes too slow. The main steps for creating SVM classifiers are the following:

1. The model starts with low-dimensional data;
2. It uses the existing data to create higher dimensions;
3. It finds a decision boundary that separates the higher dimensional data into the number of classes.

To avoid this potential problem, the decision boundary is found in higher dimensions by using *kernel functions*—this mathematical technique is called *kernel trick*. A kernel is a function K that, when evaluated on two vectors of dimension n, s and z, yields the same result as the dot product of the transformation ϕ of these two vectors into a space of higher dimension m, *i.e.*,

$$K(s,z) = \phi(s) \cdot \phi(z). \tag{2.32}$$

Kernels eliminate the need to transform the data from low to high dimensions. They use dot products between every pair of training points to compute their high-dimensional relationships and find the optimal decision boundary.

Two of the most popular kernel functions are the *polynomial kernel* and the *radial kernel* or *radial basis function*.

The polynomial kernel is mathematically defined as $(x_a \cdot x_b + r)^d$, where r and d are parameters determining the coefficient and the degree of the polynomial, respectively. The optimal values for these parameters are found using CV.

The radial kernel (or radial basis function) is obtained from the polynomial kernel by taking $r = 0$ and $d \to \infty$. It has the form $e^{\gamma(x_a - x_b)}$. Similarly for the polynomial kernel, the optimal value for the *gamma* parameter is obtained by CV.

When applied to regression problems, SVMs reverse the objective: instead of maximising the margin between two classes while limiting its violations, an SVM regressor tries to fit as many instances as possible on the margin, while limiting violations (*i.e.*, instances off the margin). Adding training instances within the margin does not affect the model's predictions. Non-linear cases are handled in a similar way to the classifier, by applying the kernel trick.

Under the hood, SVMs involve several beautiful equations, which are, however, outside the scope of this thesis. We refer the reader to Vapnik (2000) and Géron (2017) for a detailed description on how SVMs make predictions and how their algorithms work.

2.2.6 Decision Trees

A *decision tree* (DT) is a supervised ML method that stratifies or segments the predictor space into simpler regions. A tree is built by recursively partition the training set, by creating one or nesting more `if-then` statements for the predictors. These methods get their name from the fact that the set of splitting statements used to stratify the feature space can be summarised and represented in a tree-like structure, as the one illustrated in fig. 2.4.

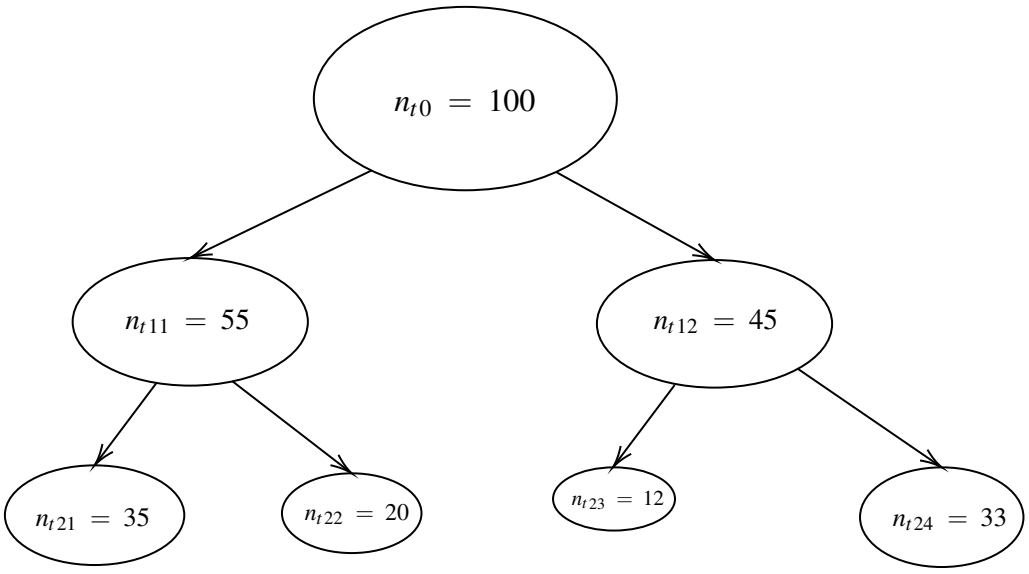

Figure 2.4: An example of a decision tree. Every node t is associated with n_t observations from the data set.

To obtain a prediction for a new observation, the model follows the `if-then` statements defined by the tree using values of the instance's predictors, until it reaches a terminal *node* or end *leaf*. For a given sample, the `if-then` statements generated by a tree define an unique path to a single leaf. A *rule* is a set of `if-then` statements that have been compiled into independent conditions (Kuhn, Johnson et al., 2013). A very simple example of a tree-based model, inspired in the tree depicted in fig. 2.4, is written as the set of conditions of the left-hand side of fig. 2.5. The corresponding

rules in the region space, together with the predicted values, are depicted on the right-hand side graph of the same figure. Four rules can be extracted from the tree of fig. 2.5:

start;
if *predictor A* ≥ 3 **then**
  if *predictor B* ≤ 12 **then**
    outcome $\leftarrow 3.14$;
  else
    outcome $\leftarrow 2.72$;
  end
else
  if *predictor B* ≤ 8 **then**
    outcome $\leftarrow 0.0073$;
  else
    outcome $\leftarrow 0.0123$;
  end
end

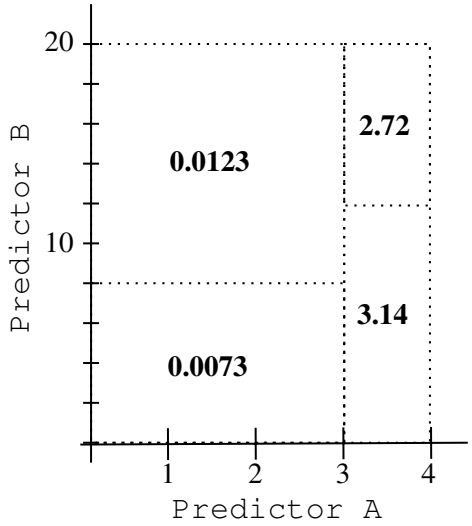

Figure 2.5: Set of `if-then` statements (*left*) and corresponding regions (*right*) defined by a simple tree-based model. These structures are a representation of the tree of fig. 2.4.

```
if predictor A >= 3 and predictor B <= 12 then outcome = 3.14
if predictor A >= 3 and predictor B >  12 then outcome = 2.72
if predictor A <  3 and predictor B <= 8  then outcome = 0.0073
if predictor A <  3 and predictor B >  8  then outcome = 0.0123
```

The goal is to partition the data into smaller, more homogeneous subsets. Starting at the tree root, data are split on the predictor and for the value that results in the largest *information gain* (IG), $\mathscr{I}_g$. The latter is defined as follows (Raschka and Mirjalili, 2017):

$$\mathscr{I}_g(DS_P, x_i) = I(DS_P) - \sum_{j=1}^{m} \frac{n_j}{n_P} I(DS_j), \tag{2.33}$$

where DS_P and DS_j are the subsets of the parent and j^{th} child node, respectively, x_i is the feature to perform the split, I is the impurity measure,* m is the number of child nodes, and n_P and n_j are the number of training instances at the parent and j^{th} child node, respectively.† Maximising the IG is equivalent to minimise the impurity in the child nodes.

The two most common impurity measures or splitting criteria used in classification trees are the *entropy*, I_H, and the *Gini index*, I_G.

For all non-empty classes, the entropy is defined by

$$I_H(t) = -\sum_{i=1}^{c} p(i \mid t) \log_2 p(i \mid t), \tag{2.34}$$

* In this context, *purity* is a measure of homogeneity. The less labels are in a node (classification context) or the closest the numbers in the node (regression case), the more homogenous (pure) it is.

† Every node t of a decision tree is associated with a set of n_t instances or data points from the training set.

where $p(i \mid t)$, the proportion of observations belonging to class i for a node t, has to respect the condition $p(i \mid t) \neq 0$. At a given node, the entropy is zero when all its observations belong to the same class, and it is maximal when the distribution of classes in the node is uniform.

The Gini index is defined as

$$I_G(t) = \sum_{i=1}^{c} p(i \mid t)[1 - p(i \mid t)] = 1 - \sum_{i=1}^{c} p(i \mid t)^2. \tag{2.35}$$

It is a criterion to minimise the probability of misclassification or the misclassification error. Similar to entropy, for a given node, the Gini index is maximal when the classes are perfectly mixed.

For regression trees, the IG is also maximised by minimising the impurity function in the child nodes. However, in this case, the latter is commonly the MSE, which is very similar to eq. (2.4), page 12:

$$I(t) = \text{MSE}(t) = \frac{1}{n_t} \sum_{i \in DS_t} (y_i - \hat{y}_i)^2, \tag{2.36}$$

where n_t is the number of examples belonging to the training subset DS_t at node t, and y_i and $\hat{y}_i$ are the true and predicted values of the response, as defined before. In the context of regression DTs, the MSE is generally known as the *within-node variance*, and the splitting criterion is commonly referred as the *variance reduction*.

The growth process of a DT, usually known as *recursive partitioning*, is repeated until some stopping criterion is satisfied—for instance, until the leaves are pure, or the within-node variance reaches a certain threshold. A prediction is produced by computing the mean or the mode of the observations contained in the terminal nodes.

Decision trees intrinsically perform feature selection in the data set, because if a variable is never used in a split, it will not be part of the prediction equation. However, this advantage is weakened in the presence of highly correlated features, since, in that case, there is a certain level of randomness in the choice of which predictor to use in a split. More predictors can be used than actually necessary, affecting variable importance by diluting it among them (Kuhn, Johnson et al., 2013).

Tree-based models have the advantage of being simple and interpretable, when the DTs are not large, and they can be quickly computed. Because they are built in a simple and logic way, they can effectively handle many types of features, even if data are missing, without the need of pre-processing. In addition, they do not require any transformation of the predictors in situations of non-linear data, nor normalisation or standardisation, because DTs analyse one feature at a time, rather than considering weighted combinations of them (Raschka and Mirjalili, 2017). However, they can be unstable (small changes in the data can affect considerably the structure of the tree and, hence, its interpretability) and, in terms of predictive performance, they typically are not comparable to other supervised ML approaches, such ensembles of trees, because they define rectangular regions that have the potential of yielding large prediction errors when the the relationship between the explanatory variables and the response cannot be adequately defined by rectangular subspaces of the predictors (Kuhn, Johnson et al., 2013).

In the following section, we will cover some important tree-based ensemble algorithms, which combine many trees into one model and tend to have better predictive performance than single DTs.

2.2.7 Ensemble Learning

Ensemble methods are ML techniques that combine different models (typically weak regressors or classifiers) into a single model having better performance than each of the individual models alone (Hastie, Tibshirani and J. Friedman, 2009). They started to be developed in the 1990s, and became popular ever since. Examples are *Bootstrap Aggregating*, *Random Forests*, and *Boosting*, just to name a few. These approaches have in common the fact that they are fully non-linear, highly flexible, and hard to interpret. They also tend to have very good predictive performance, in particular tree-based ensembles.

In the following sections, we will discuss the three aforementioned examples of ensemble methods, that use trees as building blocks to construct prediction models. Two of them, random forests and gradient boosting, because they were heavily used as workhorses during this thesis to train regression models, are going to be briefly introduced in this section and will be described in more detail in chapter 3 (section 3.2).

Bagging

Bootstrap Aggregation or *Bagging*, is a simple ensemble method proposed by Breiman (1996a) that uses bootstrapping together with any regression or classification model to build an ensemble (Kuhn, Johnson et al., 2013; Hastie, Tibshirani and J. Friedman, 2009). For every bootstrap sample, bagging trains a tree model. If we train m trees, then the predictions of each tree are averaged out to produce the bagged model's final prediction. Its simple structure is summarised in the main steps for a bagging algorithm of fig. 2.6.

for $i = 1$ *to* m **do**
    Generate bootstrap sample of the orginal data;
    Train unpruned tree model on this sample;
end
outcome $\leftarrow$ average of m individual predictions;

Figure 2.6: Main ideas for a bagging algorithm (adapted from Kuhn, Johnson et al., 2013).

Due to their aggregation process, bagged models reduce the variance of the predictions when compared to simple DTs, making them more stable. This is particularly notorious in regression trees. However, (*i*) they can be computationally expensive, especially if the learners are not parallelised by the modeler, (*ii*) they are less interpretable than simple trees, and (*iii*) they do not reduce variance as effectively as other ensemble methods, such as *random forests* (RFs), mainly because the trees in bagging are not completely independent of each other, since all predictors are considered at every split of any tree via the bootstrap sampling (Hastie, Tibshirani and J. Friedman, 2009).

The logical step to improve bagging is to reduce the correlation among trees, which from a statistical point of view can be achieved by adding randomness to the building process of the trees. That was proposed by Breiman (2001) with the well-known and successful *random forests* algorithm, which we will briefly address in the following section.

Random Forests

Random forests are among the most popular ML methods, thanks to their good predictive performance, robustness, and ease of use (Torgo, 2011). The algorithm is part of the ensemble learning family, since it trains several DTs when learning a model from a data set.

The general principle is analogous to bagging, in the sense that a RF is a set of tree-based models, each built from a distinct bootstrap sample of the training data set. Similarly to DTs, each node in a tree of the ensemble corresponds to a condition on a single feature or variable. That condition is designed to split the data into two sets, so that similar response values end up in a same given new node. Differently from bagging, in order to increase the variability of the individual models that make up the ensemble (forest), instead of using the whole set of p variables, each tree is grown using a random set of $m < p$ features, which is different in every node, so that only a subset of predictors is used to compare the possible splits and choose the one that leads to the best result. This prevents a strong predictor to dominate the top splits and increases the variety among trees, decorrelating them.

Random forests are easily adaptable and versatile methods, being widely used in the context of classification and regression problems. In essence, the ensembles are built in a similar way in both cases. The main differences are (*a*) the type of DTs used to build the forest is different in each case (classification trees *vs*. regression trees); (*b*) different criteria are used to grow the individual trees (Gini impurity measure or entropy in a classification task, and MSE in a regression problem); and (*c*) in classification, the predictions by all the trees are aggregated by assigning the label obtained by *majority vote*, while in regression the predicted target value is calculated by averaging the predictions over all DTs.

Given their robustness, performance, and easy of use, we selected RFs as one of the two ML tools to tackle our research questions. They will be covered in more detail in chapter 3, in particular in section 3.2.2 (page 49).

Boosting Methods

Boosting is currently one of the most powerful techniques used in ML (Hastie, Tibshirani and J. Friedman, 2009). In boosting (Schapire, 1990; Freund and Schapire, 1999; Freund, 1995; Schapire, 2003), the ensemble consists of simple base models, known as *weak learners*, such as *decision stumps*,* which often perform only slightly better than a baseline learner or a random guess. The main idea behind boosting is to focus on the training instances that yield a bad performance, so

* A *decision stump* is a ML model consisting of a one-level tree, *i.e.*, a decision tree containing only a single variable or attribute for splitting, with the root (the first node) immediately connected to terminal leaves (Géron, 2017; Kuhn, Johnson et al., 2013).

that an ensemble is incrementally built by adjusting the weights of the examples according to the error of the previous prediction (Raschka and Mirjalili, 2017). Weak learners are sequentially built by minimising the errors obtained from previous iterations, while increasing (or boosting) the influence of high-performing models. By allowing the weak learners to iteratively learn from the training examples that were misclassified or yielded bad predictions, the performance of the ensemble is improved (or boosted) as a whole. Similarly to RFs, boosting can lead to a decrease in both the bias and variance (Breiman, 1996b).

The main ideas of the fundamental formulation of the boosting algorithm can be summarised as follows:

1. A subset of instances is randomly sampled from the training data set without replacement, to train a weak learner.
2. A second subset of examples is randomly sampled without replacement and added typically to 50 % of the instances that previously lead to bad predictions in order to train another weak learner.
3. The examples which the previous weak learners disagree upon are sought in the training data set and used to train a third weak learner.
4. The previous weak learners are combined by means of some rule, such as majority voting in the case of classification.

Boosting and RF are cousin algorithms, in the sense both train several decision trees for a given data set. The main difference between them is that in RFs trees are independent, while in boosting each tree focus its learning on the loss of the previous modelled tree.

Developments to the original algorithm, such as *AdaBoost* or *adaptive boosting* (AB) and *gradient boosting* (GB), use the complete training data set to feed the weak learners.

Instead of subsampling the training data set, AdaBoost (Freund and Schapire, 1997) sequentially re-weights the cases in each iteration to build a strong model able to learn from the mistakes of the weak learners used in the previous iterations. In each iteration, the algorithm focuses more and more on the hard cases, *i.e.*, those training examples that were underfitted, so that the weak learners sequentially improve on top of their predecessors by correcting them.

Similarly to AB, gradient boosting (J. H. Friedman, 2001) focus on adding weak learners sequentially to the ensemble, each one correcting its predecessor. The main difference to AdaBoost, however, is that *gradient boosting* (GB) tries to fit the new weak learners to the residuals yielded by the previous learners, instead of tweaking the examples' weights at every iteration (Géron, 2017). In order to do that, GB employs a gradient descent algorithm, so that errors are minimised in the sequential models.

Currently, the most popular development of GB is *extreme gradient boosting* (XGBoost). The majority of the challenges on the 2015 Kaggle ML competition were won using XGBoost, and the top 10 placements at the KDD Cup 2016 ML competition used XGBoost. XGboost is a decision tree based ensemble algorithm built upon the GB framework. It optimises GB by performing parallel processing and tree-pruning, it uses regularisation to avoid both bias and overfitting, and it is capable of handling missing values (T. Chen and Guestrin, 2016). Given its robustness, speed,

and performance, we selected XGBoost as the other ML technique (in addition to RFs) to address our research questions. We will give a more detailed account of GB and XGBoost in section 3.2.3 (page 51).

Artificial Neural Networks

Tree-based ensemble algorithms typically yield top results when dealing with structured or tabular data. Other types of non-linear algorithms, such as *artificial neural networks* (ANNs), tend to outperform all other frameworks in prediction problems involving unstructured data, such as images, text, time series, and others. While neural networks are popular, powerful, versatile, and scalable ML techniques, that can be used both on classification and regression tasks, their study is out of the scope of this thesis because, on the one hand, they typically excel when used with unstructured data and, as we will see in chapter 3, our data is structured in tabular form; on the other hand, they are usually more computationally expensive than traditional algorithms, and we are focused in building a computationally cheap tool—state of the art deep learning algorithms can take several weeks to train completely from scratch, depending on the data, and the depth and complexity of the neural network. The interested reader is referred to the original article from Bishop et al. (1995), and the books from Ripley (2007), Titterington (2010), Torgo (2011) and Kuhn, Johnson et al. (2013).

2.3 Useful Astronomical Concepts

In this section, we will briefly describe some astronomical concepts that are useful to contextualise the problem we are trying to solve in this project and the variables related to it.

Stellar classification. All information about the physical properties of stars comes directly or indirectly from their spectra, which are obtained when the electromagnetic radiation from a star is analysed by splitting it with a prism or diffraction grating. In particular, by studying the absorption lines and their strength, stellar composition, temperature, and mass can be deduced.

In Astronomy, stars are commonly classified according to their spectral characteristics. The strengths of the spectral lines vary mainly due to the temperature of the photosphere of the star. The spectral classification currently in use was developed at the Harvard observatory, in the beginning of the 20^{th} century (see *e.g.*, Karttunen et al., 2007). The main stellar *spectral types* are denoted by capital letters, ordered according to the mean surface temperature of the star. With the temperature decreasing to the right, the *early classes* are

O–B–A–F–G–K–M.

For example, type O stars are blue, with surface temperatures of 20 000 K to 35 000 K, and they are very massive, from about 16 up to more than 100 Solar masses; G stars, like the Sun, are yellow and with a surface temperature of approximately 5500 K; K stars are orange-yellow, with a surface

temperature of about 4000 K; and M stars are red, with a surface temperature of about 3000 K, and their masses are significantly lower than that of the Sun.

Rotation of stars. Stars are rotating bodies, with periods ranging from a fraction of a second, such as neutron stars, to tens of days (see *e.g.*, Carroll and Ostlie, 2017). It is known that stellar rotation is a key phenomenon for the generation of magnetic fields and the transfer of angular momentum of solar type stars.

Stars are known to decrease their rotation angular speed as they age and loose angular momentum. Consequently, rotation periods can be used to infer the age of a star, in a process known as *gyrochronology* (Santos et al., 2019). Rotation periods are typically measured by detecting magnetic phenomena at the stellar photosphere, such as *stellar spots*. Sun spots have been regularly recorded since the 17$^{\text{th}}$ century, in a well known time series (see fig. 2.7),*, which has made it possible to measure the solar activity over the last centuries. The same principle is applied to other stars to measure their surface activity.

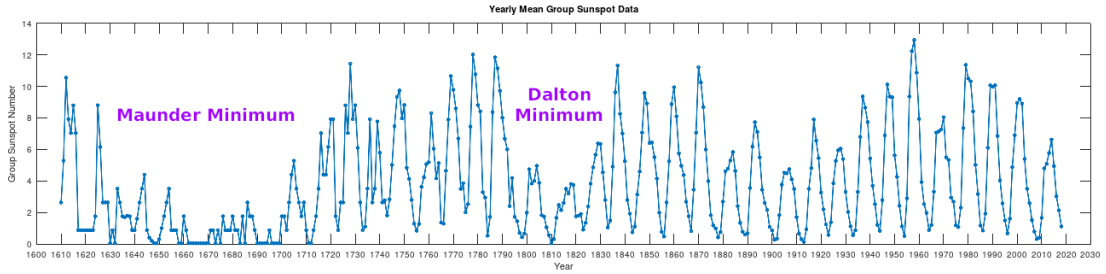

Figure 2.7: Yearly mean number of groups of sunspots between the years 1610 and 2018. Visible are two periods of lesser solar activity, known as the Maunder and the Dalton minima. The data was retrieved from the SILSO website (`http://sidc.be/silso/`).

The FliPer metric. The *FliPer* or *Flicker in Power* metric (Bugnet et al., 2018) is a measure of the total *power spectral density* (PSD) of a star between a low frequency cut-off, ν_C, and the Nyquist frequency, ν_N:

$$\text{FliPer}(\nu_C) = \frac{1}{\nu_N - \nu_C} \int_{\nu_C}^{\nu_N} \text{PSD}\, d\nu - P_n, \tag{2.37}$$

where P_n is an estimate of the photon noise level.† The FliPer metric is correlated with the *stellar surface gravity*, $\log g$. The latter corresponds to the acceleration of gravity experienced at the surface of a star, at the equator, including the effects of rotation (*e.g.*, Stahler and Palla, 2008).

The photometric activity index. The *photometric activity index*, S_{ph}, corresponds to the standard deviation calculated over segments of the light curve of width $5 \times P_{\text{rot}}$ (Mathur, García et al., 2014). In this thesis, S_{ph} corresponds to the average of those segments' values, and for that reason, we will be referring to it as the *photometric activity proxy* (Breton et al., 2021).

* A time series is a collection of data indexed in time (Skumanich, 1972).

† *Photon noise* is the inherent natural variation of the light or photon flux incident on a detector (*e.g.*, Hasinoff, 2014). It depends on the magnitude of the stellar object.

Traditional methods to measure rotation periods of stars. Stellar surface rotation periods can be estimated using several methods. The most commonly used during the last decade are the *periodogram analysis* (Reinhold, Reiners and Basri, 2013), *Gaussian processes* (Angus, Morton et al., 2018), *gradient power spectrum analysis* (Shapiro et al., 2020), *autocorrelation function* (McQuillan, Mazeh and Aigrain, 2014), *time-period analysis based on wavelets* (García et al., 2014), and a combination of different diagnostics such as the *composite spectrum* (Santos et al., 2019). Of interest for this work are (*a*) the time-period analysis using a wavelet decomposition, (*b*) the *autocorrelation function* (ACF) of the light curve, and (*c*) the *composite spectrum* (CS).

In the time-period analysis, wavelets of different periods are cross-correlated with the stellar light curve, to obtain the *wavelet power spectrum* (WPS). The latter is projected over the period-axis, to obtain the one-dimensional *global wavelet power spectrum* (GWPS). Several Gaussian functions are fitted to the GWPS, starting with the one with the highest amplitude. The Gaussian and corresponding peak are removed, and the next highest Gaussian is fitted in an interactive way, until no peaks are left above the noise level. The peak of the highest fitted peak in the GWPS is assigned as the rotation period recovered by this methodology: $p_{\rm rot}^{\rm gwps}$. The period uncertainty is taken as the half width at half maximum (HWHM) of the Gaussian function (Breton et al., 2021).

In the ACF of the light curve method, the rotation period, $p_{\rm rot}^{\rm acf}$, is taken as the period of the highest peak in the ACF at a lag greater than zero (T. Ceillier et al., 2017).

In the CS method, the product between the normalised GWPS and the ACF, *i.e.*, the *composite spectrum*, is calculated. This operation aims at amplifying the peaks present in both the GWPS and the ACF, which possibly will be related to the rotation of the star, while the signals appearing only in one of the two methods (possibly due to instrumental effects, having a different manifestation in the two analysis) will be attenuated. Several Gaussian functions are fitted to the CS, in an iterative process similar to the one described for the GWPS. The extracted rotation period, $p_{\rm rot}^{\rm cs}$, will correspond to the period of the first Gaussian function being fitted (the one with the highest amplitude). The uncertainty is estimated with the HWHM of the Gaussian function (Breton et al., 2021).

These and other variables related to the three aforementioned methods are detailed in table A.1.

CHAPTER 3

Materials and Methods

Torture the data, and it will confess to nothing.
— Ronald Coase (1910–2013)

A shortcut is the longest distance between two points.
— Anonymous

Contents

THE MAIN GOAL of this project is to develop robust yet computationally cheap models from tabular astronomical data for predicting stellar rotation periods of K and M stars from the *Kepler* catalogue. To accomplish this, specific materials and methods have to be used to build the models and assess their performance. We will be addressing them in this chapter. We start by describing the data that served as the basis to build the sets from which we trained the regression models, and explain the reasons behind the choice of that real-world data set as starting point for our research. Then, we will define the predictive task, in which rotation periods

of stars are framed as a regression problem, and will formalise the two ensemble methods selected to tackle the problem: random forests, and gradient boosting. Next, we describe the experimental design employed in the training and evaluation of the performance of the models, and finish with the description of the methods used to compare the different results.

3.1 Materials

The goal of this thesis is to develop an efficient lightweight ML tool to automatically predict reliable stellar rotation periods of *Kepler* K and M objects, resorting on a set of tabular *standard* predictors. In this context, "standard" means that they are commonly obtained directly or indirectly from astronomical observations.

We started by analysing structured data from the *Kepler* catalogue related to K and M dwarf stars, already in tabular form, published by Santos et al. (2019) and Breton et al. (2021). Both catalogues—hereafter referred as S19 and B21, respectively—contain all the predictors and targets we considered in our work. B21 was used as the source of most of the predictors. S19 was used specifically to extract the rotation periods and features that are directly obtained from *Kepler* observations. The latter are commonly known as "stellar parameters" in the astronomical community. Examples are the mass of the star and its effective temperature, just to name a few. However, we will avoid using the word "parameter" in this context, in order to prevent confusion with *model parameter*. Instead, we will adopt the terminology *astrophysical variables* when referring to them.

B21 is available in tabular form, without any particular clustering or classification of the variables. However, we decided to reorganise the predictors according to their nature and/or method by which they were obtained. We identified six groups, with the following characteristics:*

1. *Rotation periods* (Prot, `prot`) — rotation periods obtained by combining the ACF, CS, and GWPS time-period analysis methods with the KEPSEISMIC light curves, for the 20-, 55-, and 80-day filters: $P_{\rm rot}^{\rm ACF}$ (`prot_acf_xx`), $P_{\rm rot}^{\rm CS}$ (`prot_cs_xx`), and $P_{\rm rot}^{\rm GWPS}$ (`prot_gwps_xx`), where *xx* stands for each of the aforementioned filters.†
2. *Astrophysical* (Astro, `astro`) — predictors related to the physical properties of the stars, and that are obtained directly or indirectly from the observation, to wit:
   - effective temperature, $T_{\rm eff}$ (`teff`), and its corresponding upper and lower errors, $T_{\rm eff}^{\rm err,\,up}$ (`teff_eup`) and $T_{\rm eff}^{\rm err,\,low}$ (`teff_elo`) respectively;
   - the logarithm of surface gravity, $\log g$ (`logg`) and its upper and lower error limits, $\log g^{\rm err,\,up}$ (`logg_eup`) and $\log g^{\rm err,\,low}$ (`logg_elo`);
   - the mass of the star, M (`m`), and its upper and lower errors, $M^{\rm err,\,up}$ (`m_eup`) and $M^{\rm err,\,low}$ (`m_elo`);
   - the magnitude from the *Kepler* input catalogue, $K_{\rm p}$ (`kepmag`); and

* Words between parentheses, written in `typewriter font`, refer to the names of the variables as they stand in table A.1 and in the data frames.

† KEPSEISMIC are time series optimised for asteroseismology. They are available at the Mikulski Archive for Space Telescopes (MAST), via the link https://dx.doi.org/10.17090/t9-mrpw-gc07.

- the Flicker in Power or *FliPer* values, $F_{0.7}$ (`f_07`), F_7 (`f_7`), F_{20} (`f_20`), and F_{50} (`f_50`), respectively corresponding to cut-off frequencies of 0.7, 7, 20 and 50 µHz.*

3. *Time Series* (TS, `ts`) — quantities that are related to the properties of the time series:
   - length of the light curve in days, l (`length`);
   - the start and end time of the light curve, S_t (`start_time`) and E_t (`end_time`) respectively;
   - the bad quarter flag, Q_{bad} (`bad_q_flag`);
   - the number of bad quarters in the light curve, nQ_{bad} (`n_bad_q`);
   - the photometric activity proxy or index, S_{ph} (`sph`), and its error, S_{ph}^{err} (`sph_e`), as provided by Santos et al. (2019);
   - S_{ph}^{ACF}, computed from the ACF method to obtain the rotation period (`sph_acf_xx`), as provided by Breton et al. (2021), and its corresponding error, $S_{ph}^{ACF,\,err}$ (`sph_acf_err_xx`);†
   - the height of P_{ACF}, G_{ACF}, the period of the highest peak in the ACF at a lag greater than zero (`g_acf_xx`), for each of the *xx*-day filter; and
   - the mean difference between the height of P_{ACF} at the two local minima on both sides of P_{ACF}, H_{ACF} (`h_acf_xx`), for each of the aforementioned filters.
4. *Global Wavelet Power Spectrum* (GWPS, `gwps`) — quantities obtained after a time-period analysis of the light curve using a wavelet decomposition:
   - amplitude (`gwps_gauss_1_j_xx`), central period (`gwps_gauss_2_j_xx`), and standard deviation (`gwps_gauss_3_j_xx`) of the j^{th} Gaussian fitted in the GWPS with the *xx*-day filter, where *xx* stands for 20, 55, or 80 days;
   - the mean level of noise of the Gaussian functions fitted to the GWPS for the *xx*-day filter (`gwps_noise_xx`);
   - the χ^2 of the fit of the Gaussian function on GWPS for the *xx*-day filter (`gwps_chiq_xx`);
   - the number of Gaussian functions fitted to each GWPS for the *xx*-day filter (`gwps_n_fit_xx`); and
   - S_{ph}^{GWPS} calculated from the GWPS method (`sph_gwps_xx`), as provided by Breton et al. (2021), and its corresponding error, $S_{ph}^{GWPS,\,err}$ (`sph_gwps_err_xx`).
5. *Composite Spectrum* (CS, `cs`) — variables obtained from the product between the normalised GWPS and the ACF:‡
   - amplitude of P_{CS} (the period of the fitted Gaussian of highest amplitude), H_{CS} (`h_cs_xx`), for the *xx*-day filter;

* FliPer is a measure of the total power in the power spectral density of the star between the cut-off frequency, ν_C, and the Nyquist frequency, ν_N (recall section 2.3, page 32).

† In Breton et al. (2021), S_{ph} is split in several values, each corresponding to the process from which it was obtained (ACF, CS, or GWPS) and to the filter applied to the light curves (20, 55, or 80 days).

‡ The composite spectrum (CS) is computed in order to amplify the peaks present in both the GWPS and the ACF, and to attenuate the signals appearing in one of them resulting, possibly, from instrumental effects that have a different manifestation in each analysis (recall section 2.3, page 33).

- the mean level of noise of the Gaussian functions fitted to the CS, cs_{noise} (`cs_noise_xx`), for the *xx*-day filter;
- the χ^2 of the fit of the Gaussian function on the CS for the *xx*-day filter (`cs_chiq_xx`);
- $S_{\text{ph}}^{\text{CS}}$ computed from the CS method (`sph_cs_xx`), and its error, $S_{\text{ph}}^{\text{CS, err}}$ (`sph_cs_err_xx`) for the *xx*-day filter; and
- the amplitude (`cs_gauss_1_j_xx`), central period (`cs_gauss_2_j_xx`), and the standard deviation (`cs_gauss_3_j_xx`) of the j^{th} Gaussian fitted in the CS with the *xx*-day filter, for $j \in \{1, 2, \ldots, 6\}$, where *xx* stands for 20, 55, or 80 days.

6. *Unknown* (Wav, `wav`) — variables, named `wav_scl_{max, min}_xx` in the data set, which are not referenced in Breton et al. (2021) and, hence, whose nature is unknown.

The list of explanatory variables, as they appear in the final data set, after exploratory data analysis, cleaning, and engineering, is referenced in table A.1.

An alternative approach to real-world data would have been to use synthetic data to train the models. These data would be obtained from reliable stellar models, such as the PLATO Simulator (Marcos-Arenal et al., 2014), so that we could have full control over the characteristics of the stars and related variables. In addition, by using synthetic data would have allowed us to prevent potential bias introduced by the predictors, given that many of the canonical variables used to estimate the reference stellar rotation periods, such as the *amplitude of the Gaussian* fitted in the GWPS, are derived throughout the processes that culminate in the estimation of the final value. They are not *observables*, *i.e.*, they are not directly measured through instruments, nor are extracted directly from the light curves. They are derived features, whose values are strongly correlated with the reference ("true") rotation periods.

Nevertheless, we started by using previously published real data because, on the one hand, it is readily available, which is an advantage within the time-frame of this Master's thesis; and, on the other hand, we wanted to benchmark distinct machine learning approaches using different sets of predictors—thus analysing how much the predictions are affected by the variables closely connected to the method used to estimate the reference rotation periods—and compare the results with the ones obtained by Breton et al. (2021)—the results can only be comparable if data sets are built from the same reference set of observations.

In the following section, we will describe the steps we applied to S19 and B21 to build a master data set and several subsets to be used to build the ML methods with which we intended to answer the research questions.

3.1.1 Data Engineering

S19 and B21 were cleaned and merged to obtain a master data set, from which several subsets were created. During the cleaning process, (*i*) columns containing only null values were dropped, (*ii*) rows with less than 5 % of non-null variables were removed, (*iii*) a flag-type variable from S19 was dropped, (*iv*) all variable names were converted to lowercase, and (*v*) some names were changed, for clarity.

In the merging process, we selected all stars that were present simultaneously in both data sets. As we will explain in detail in section 3.2, we chose to use tree-based ensemble approaches, namely RFs and GB, to train our models. Since RFs cannot cope with missing values, we imputed `Not a Number` (`NaN`) type cells using a RF approach with 100 trees and a 0.5 sample fraction—we relied on the `missRanger` function of the **ranger** R package to accomplish this task. After the data imputation, we decided to create a parallel data set with standardised predictors for further analysis. We note that because we will be using tree-based (as opposed to distance-based) ensemble methods, normalisation of predictors is not required (Murphy, 2012; T. Chen and Guestrin, 2016). Nevertheless, because XGBoost employs a gradient descent algorithm, we decided to use the normalised version of the data set to create the subsets from which the XGBoost models would be built, in an attempt to optimise the computations during the training phase.

Several features, mainly of the CS and GWPS groups (recall group of variables in previous section, page 36), contained values with extremely large or infinite amplitudes.* They are derived variables, corresponding to some of the amplitudes, central periods, standard deviations, and mean level of noise of the Gaussians fitted to the CS and GWPS. Their extreme values may have different origins, such as instrumental errors, and possibly they could have been addressed during their creation.† In principle, their existence would not pose a problem, as there are no restrictions on extreme points both in RF and GB methods, and XGBoost can handle missing values. However, since we do not have missing values in the data sets used to train RF models, we wanted to avoid `NaN`s in the data sets created for XGBoost, so that comparisons between models created with both methods would not be affected by this aspect.

We considered several strategies to address this problem, such as imputation of extreme values by central measures, removal of outliers, or row elimination. Since we wanted to prevent depletion of the training set, we chose to replace infinite points by the non-infinite extrema present on the variable where they occur.

By analysing the distribution of examples with extreme amplitudes, we realised that those were several orders of magnitude larger (typically above 30 orders of magnitude) than the common values of the corresponding variable, whose absolute values were always below 100. An example, corresponding to the variable with the largest amplitudes for the typical values, is illustrated in fig. 3.1. The figure represents histograms and box-plots for the `gwps_gauss_3_1_80` predictor. In the bottom panels we represent the most common cases, whose values oscillate approximately between 1 and 50. From the histograms on the left hand side, we can see that most of the values are distributed within the $]0.01, 100[$ interval, and that outliers can have values above 10^{35}. By replacing infinite values by the extreme (finite) values present in the feature, we are still flagging them as very big values, without affecting the z-transform. Therefore, examples with infinite values—which correspond to less than 1 % of the total number of cases—were scaled down prior to standardisation, to avoid missing values after the z-transform. On the one hand, we are validating a strategy to predict stellar rotation periods, and it is irrelevant if the data set contains or not infinite

* This was easily seen when summarising the data after standardisation, because `NaN`s were produced out of the infinite values after the z-transform.

† A way to control them would have been to use synthetic data, obtained from reliable stellar simulators.

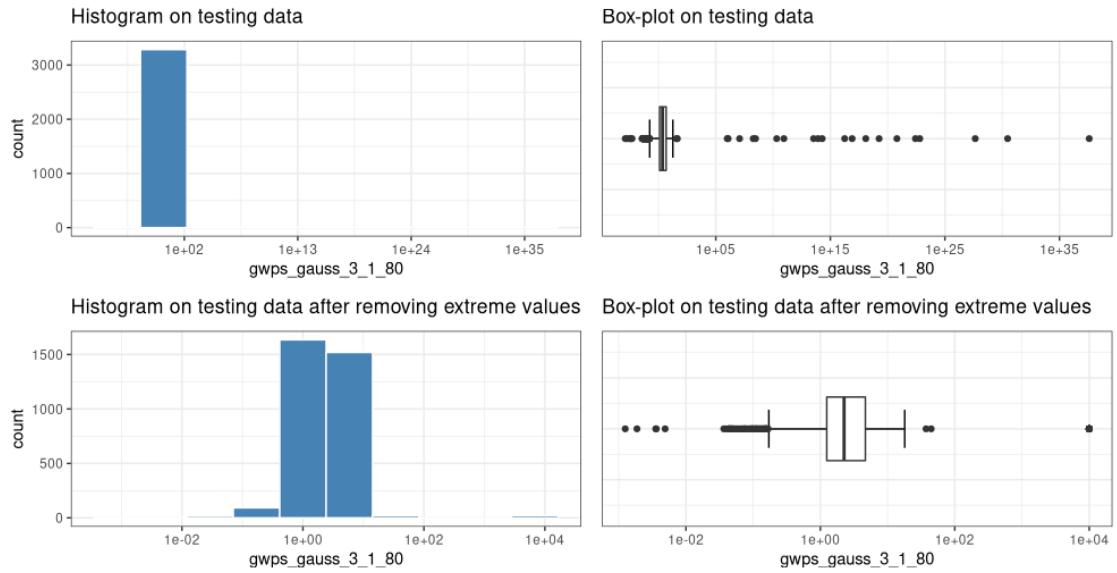

Figure 3.1: Histograms and box-plots for the `gwps_gauss_3_1_80` predictor. The histogram on the bottom row highlights the amplitude of the most common values of the variable, while the corresponding box-plot on the right still indicates the presence of outliers.

values on a methodological point of view; and, on the other hand, not only this approach is similar, in principle, to an imputation, but we believe it is important to keep large/small examples in the data set, to mimic real case scenarios, so that the model can learn from them. Performing the scaling just described will allow the model to learn from outliers, while preventing at the same time `NaN`s from appearing after standardisation, and without affecting the performance of the models.*

After the aforementioned transformations, we ended up with two data frames containing 15 006 observations and 180 predictors, one with unscaled and other with scaled predictors. These are the full or master sets, both labelled as *data set 0* (DS0).

Several subsets were created from the full data sets, containing a different number of predictors, according to their nature or the way they were extracted:†

1. The first subset was obtained by removing the nine rotation period variables (Prot group), resulting in a data set with 171 variables. This is identified as *data set 1* (DS1).
2. Another subset was obtained from the previous one, by removing the unknown Wav variables, yielding a data set with 165 predictors. This was labelled *data set 2* (DS2).
3. A third subset was created by removing the CS variables from DS1, producing a data set with 108 predictors — *data set 3* (DS3).
4. The fourth subset was obtained by removing the GWPS predictors from DS1, creating a data set with 102 explanatory variables — this is *data set 4* (DS4).

* We tested the performance of RF models with and without scaling down infinite values, using the DS3 and DS4 data sets to be defined in what follows, and no significant differences in the predictive performance were identified.

† When talking about models created with RFs or XGBoost, we will always refer to the full data set as DS0, without explicitly indicating whether we are using the normalised set or not. The type of predictors will be easily identified out of the context. The same will hold for any of the DS*n* subsets.

5. The fifth subset results from removing both the GWPS and CS variables, yielding a data set with 39 predictors — *data set 5* (DS5).
6. The sixth subset is similar to DS5, with the difference that the Wav variables were removed, remaining 33 predictors in the set — *data set 6* (DS6).
7. The seventh subset was obtained by removing all the predictors except the ones related to the time series, producing a data set with 19 explanatory variables. This was dubbed *data set 7* (DS7).
8. The last subset was obtained by removing the time series variables from DS7, resulting in a set of 14 predictors, only astrophysical in nature. This was labelled as *data set 8* (DS8).

Kepler data is divided into 90-day quarters, because the telescope had to rotate by 90° every three months in order to keep the solar panels properly aligned with the Sun and the radiator pointed towards deep space (S. Mullally, 2020). Since it is important to observe at least two cycles on an object in order to get reliable measurements, an additional data set was created from DS2, where rotation periods greater than 45 days were filtered out. This strategy also allows to avoid any problems arising from the stitching of data from different quarters. We further removed stars with surface rotation periods smaller than 7 d,* because these targets can easily be mistaken for close-in binary systems or classical pulsators, whose signals may not be consistent with stellar objects manifesting surface rotation (Breton et al., 2021). By including only stars with rotation periods between 7 d and 45 d, we were expecting to improve the predictive performance of the models. We ended up with 13 627 stellar objects, a reduction of approximately 9 % relatively to the other data sets. The number of predictors in the resulting data set, dubbed *data set 9* (DS9), was also reduced to 36: DS9 contains only the most important features identified during the learning of the RF and XGBoost models, built on top of the data sets DS0 to DS8.

A final data set, *data set 10* or DS10, was created by including only the top-10 most important variables, as evaluated by the RF and XGBoost models trained with DS9. With DS10, we wanted to infer how much the predictive performance of RF and XGBoost models would be affected by including such a reduced number of predictors which, nevertheless, would be able to explain most of the variability of the response variable. The procedure and results will be explored in sections 4.3 and 4.4.

3.1.2 Statistical Analysis of Relevant Variables

In this section, we will make use of graphical and numerical statistical tools to analyse relevant variables for our study, namely the response, and the CS, GWPS, *time series* (TS), and Astro families of features. We intend to investigate the shape of the sample distribution of the target variable, its relationship with the explanatory features, and the usefulness of the predictors to explain the variability of the response. All variables present in DS0,† including the response, are continuous. Therefore, we will explore them using histograms, box-plots, and summary statistics.

* Here and in future similar cases, "d" stands for *days*. Throughout this thesis, we will be using the International System of units.

† In this section, we will be referring to the unnormalised version of DS0, unless indicated otherwise.

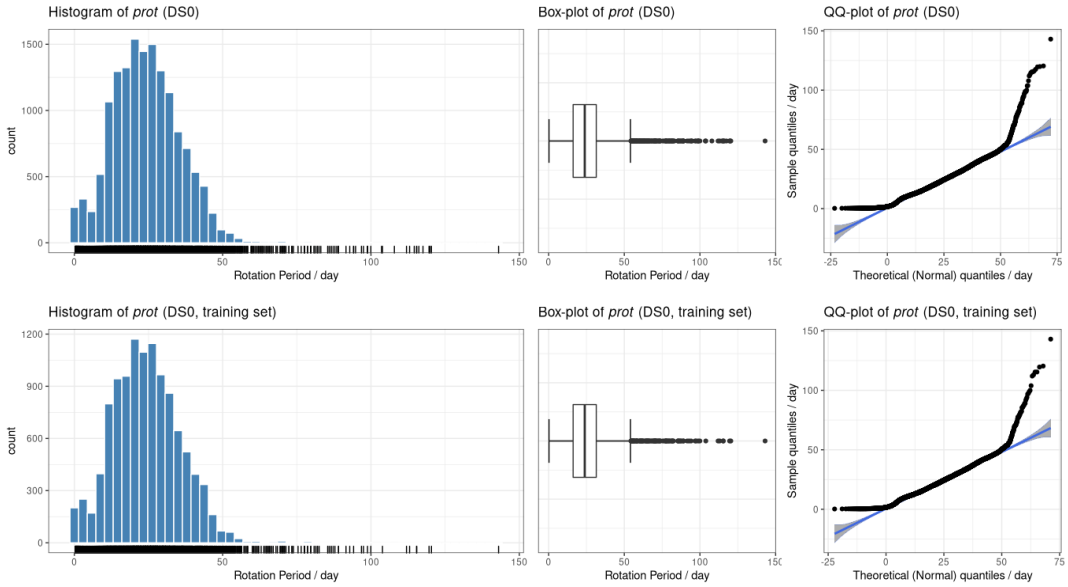

Figure 3.2: Histograms (left panels), box-plots (central panels), and QQ-plots (right panels) of the target variable, p_{rot}, on DS0 (top row) and on the training set created out of it (bottom row). The distributions are very similar to each other, right skewed, with medians equal to 23.87 d, minima and maxima equal to 0.25 d and 143.04 d, respectively. Both exhibit several outliers.

The histograms, box-plots, and QQ-plots of the target variable, p_{rot} (the stellar rotation period, in days), as extracted from DS0 (panels on top) and its training set (panels on bottom), are illustrated in fig. 3.2. The sample distributions are very similar to each other. The aforementioned graphs suggest unimodal, leptokurtic, right-skewed distributions, with several outliers towards large quantiles, corresponding to the largest stellar rotation periods, above approximately 55 d up to around 150 d. The medians are equal to 23.87 d, with minima and maxima equal to 0.25 d and 143.04 d, respectively. The fact that the means (24.43 d and 24.46 d on DS9 and its training set, respectively) are greater than the medians is a confirmation that the sample distributions are positively skewed. The histograms show that the majority of the stars in the sample have rotation periods between zero and approximately 55 d, but the right tail of the distributions extend far past this interval. Numerous points in the QQ-plot lie outside the 95 % confidence band. Therefore, there is no statistical evidence for normality in the distribution of p_{rot}.

Figure 3.3 shows the histograms, the box-plots, and the QQ-plots of the response, as extracted from DS9, both on the full data set (top panels) and on the training set (bottom panels). Similarly to DS0, the distributions on the full and training sets are very similar to each other. There is no statistical evidence for normality in both sets. The plots suggest unimodal, platykurtic, slightly right-skewed distributions. The medians are equal to 24.03 d, and the minima and maxima equal to 7 d and 44.85 d in both DS9 and its training set. A considerable number of points in the QQ-plot lie outside the 95 % confidence band, and the graph has two pronounced tails. From the histograms we can also observe that the sample lacks stars with rotation periods up to about 14 d and above 28 d, approximately. The fact that the distribution of p_{rot} is not uniform affects the learning of the models, especially in the less represented regions.

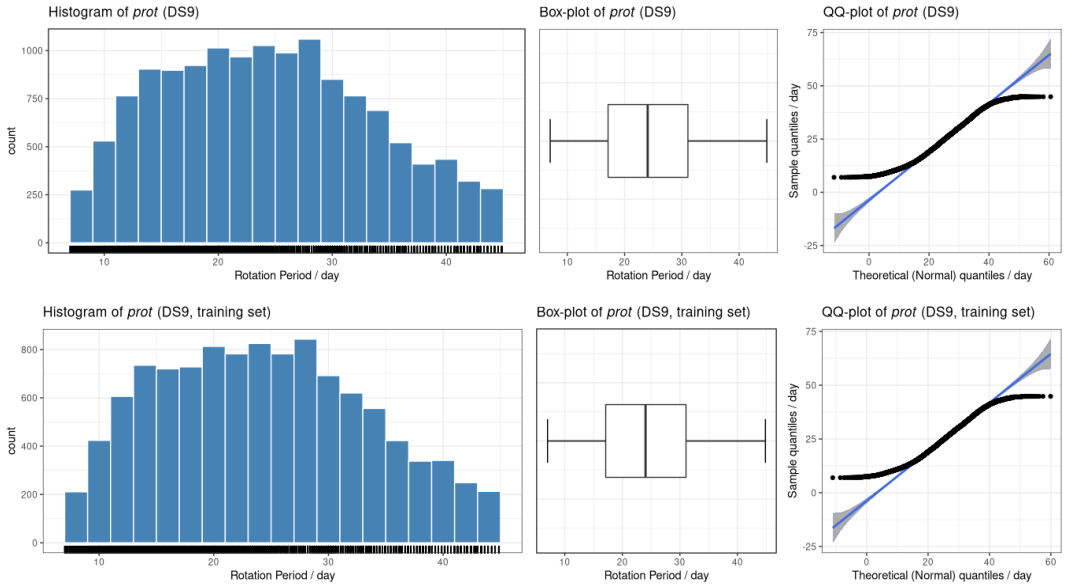

Figure 3.3: Similar to fig. 3.2, but for the DS9 data set. The distributions are very similar, approximately symmetric, with means equal to 24.44 d and 24.43 d, and standard deviations equal to 9.11 d and 9.07 d, on DS9 and its training set, respectively.

Figures 3.4 to 3.7 display the correlations between the variables of each of the families of variables introduced in the beginning of this section (page 36) and the target variable.

As expected, most Prot variables are strongly positively correlated with the response (fig. 3.4, left panel), because the latter is obtained directly from estimations of the former. This is the motivation for creating the DS1 data set. The Wav variables are completely unrelated to the target variable (fig. 3.4, right panel), and that is the reason for creating DS2 and DS6 data sets.

Some of the TS features are negatively correlated with the response (fig. 3.5, left panel), such as `sph`, `h_acf_20`, or `g_acf_20`, and we expect they will contribute for the building of models as important variables. All the variables extracted out of the ACF are strongly correlated with each other, which in principle is not a problem because we will be dealing with tree-based ensembles (see section 3.2). Apart from the mass, some cut-off frequencies of the Flicker in Power, and the effective temperature, classical Astro variables are not correlated to the response (fig. 3.5, right panel) and, hence, we are not expecting that these predictors will be very important for the learning process of models.

Concerning the CS family of variables, most do not exhibit correlation with the response, but some do, either strong positive or strong negative correlation (fig. 3.6). Examples are `h_cs_20`, `cs_gauss_3_1_20`, and `cs_gauss_2_1_20`, just to name a few, and we are expecting these to be very important during the training of the models.

Finally, for the GWPS family of features, and similarly to the CS group, several variables are strongly correlated with the response, either positively or negatively. Examples are `sph_gwps` and `gwps_gauss_2_1_20`, and we expect these to become important for the training of the models.

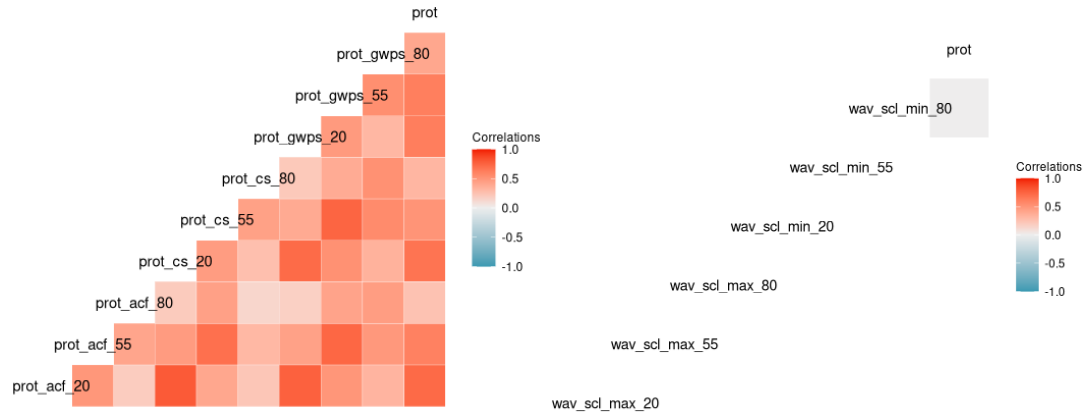

Figure 3.4: Correlations between the Prot variables and the response (left panel) and the Wav features and the target variable (right panel). Red indicates positive correlation and blue negative correlation between any two features. White colour indicates no correlation between the variables.

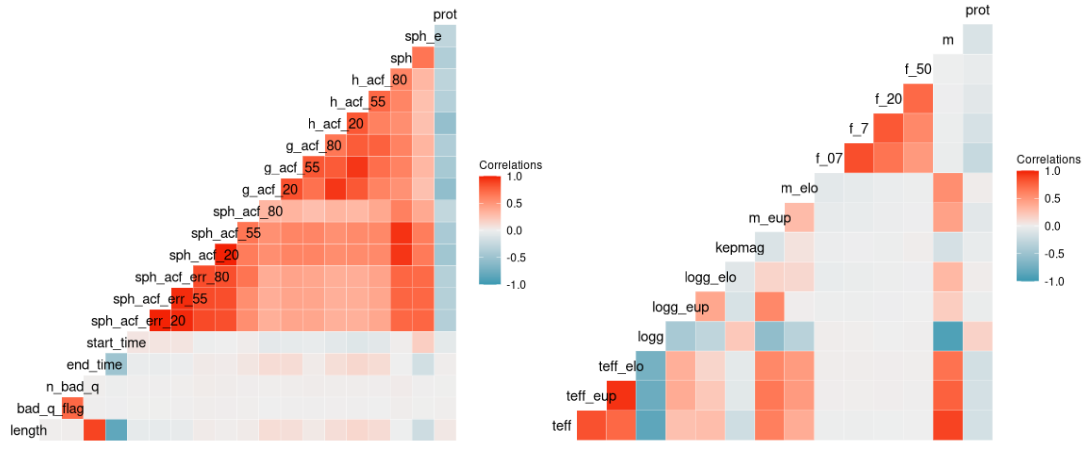

Figure 3.5: Similar to fig. 3.4, but for the TS and Astro families of variables. Several TS variables are negatively correlated with the response, while just a few Astro features exhibit (a weak) correlation with the target variable.

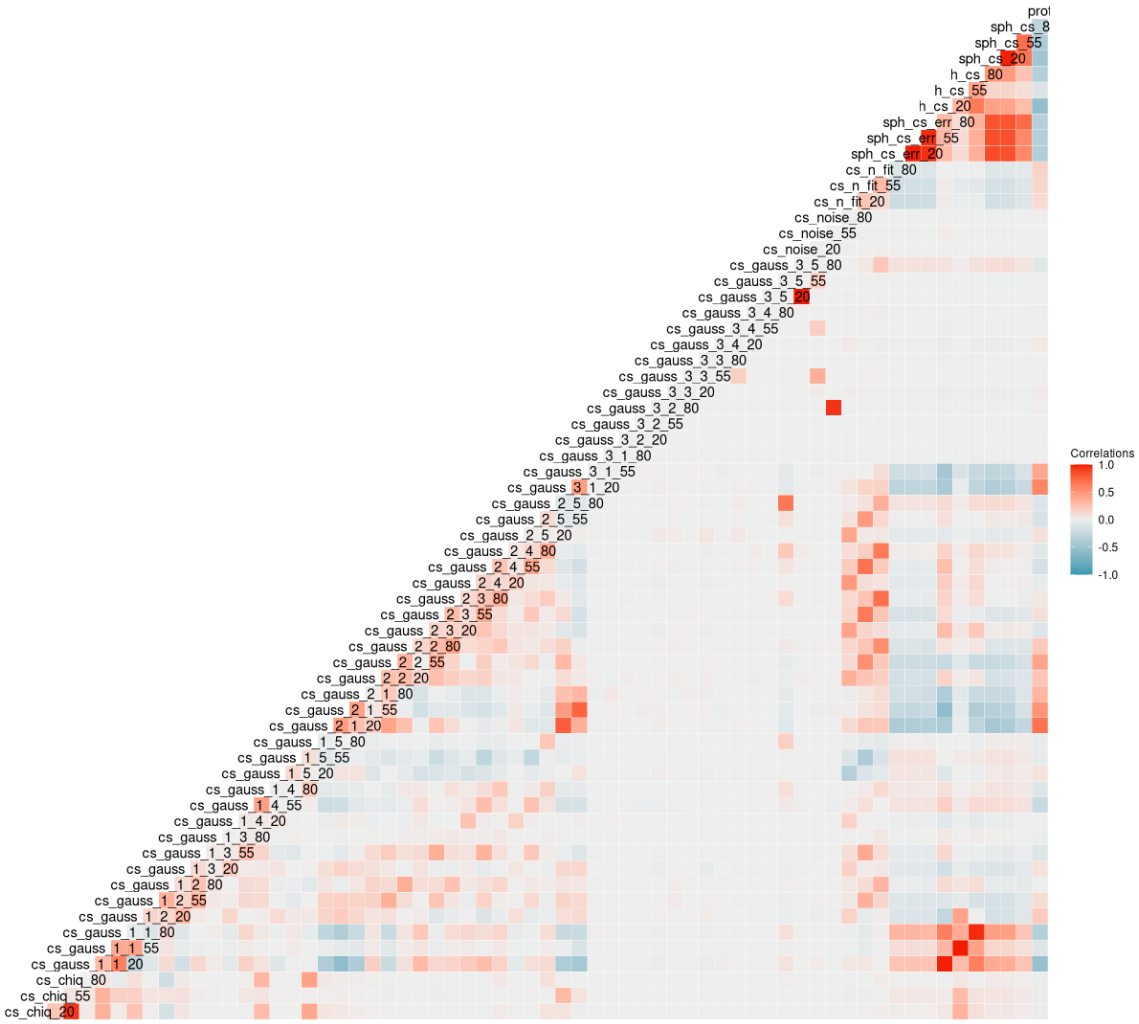

Figure 3.6: Similar to figs. 3.4 and 3.5, but for the CS family of variables. Several features have a strong either positive or negative correlation with the response, but most of the predictors exhibit a negligible correlation with it.

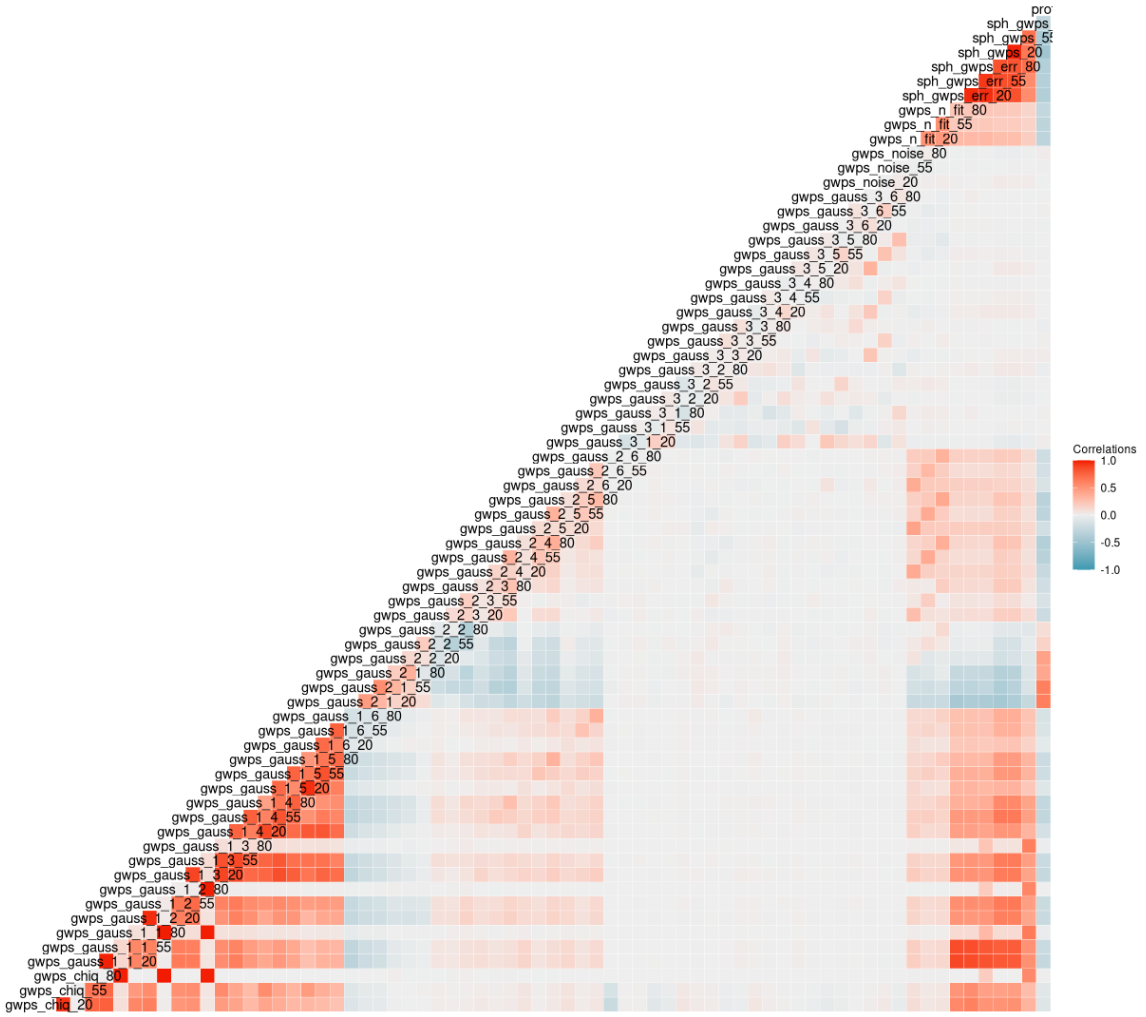

Figure 3.7: Similar to figs. 3.4 to 3.6, but for the GWPS family of variables. Several features exhibit a strong level of correlation, either positive or negative, with the target variable, while others evince a negligible correlation with it.

As we will see in section 3.2, we will be using tree-based models, whose performance, by nature, is not affected by the inclusion of highly correlated features. However, interpretability tools, such as importance estimations, are hampered by collinearity and multicollinearity.

When two features are perfectly correlated, they are likely to be chosen by any of the two algorithms. While in random forests this random choice is done for each tree (because all trees in the ensemble are independent), in boosting a link between a predictor and the response will remain stable as soon as it is learnt by the model, and thus the algorithm will stick to one of the correlated variables (but not both). Therefore, apart from small differences introduced by the model parameters, in random forests trees will chose each variable approximately 50 % of the time, and so the importance of the information contained in each of them—which is the same, because they are perfectly correlated—will be diluted, and we might not be able to realise how important are those features to predict the target variable. This situation worsens when the number of correlated variables increases. In boosting, because the algorithm sticks to one of the variables among all correlated feature set, we will be able to understand that one of the variables will have an important role in generating predictions, but we will not realise that the other variable is also important in the link between the observations and the response unless we perform a correlation analysis of the features (T. Chen, Benesty and He, 2018).

Typically, the importance profile for gradient boosting has a steeper slope than the one for random forests, since the trees from boosting are interdependent and, thus, will have correlated structures as the gradient evolves. Consequently, several predictors will be selected across the trees, increasing their importance. It is then natural that RF and GB models report differences in order and importance magnitude between features, which can be used to get further insight on the relationships between the predictors and the target variable (Kuhn, Johnson et al., 2013).

Since one of the goals of this thesis was to compare the performance of the models generated from data sets DS0 to DS10 to the Breton et al. (2021) model, we decided not to perform feature selection when training the RF learners. When training XGBoost models, however, we performed a simple univariate filtering of features by cross-validating the predictors to the response variable, in order to remove redundant or non-informative variables from the model. The procedure will be detailed in section 4.2.

In the following section, we will formalise the predictive task we set out to solve, and will describe in detail the two ensemble techniques—RFs and GB—we used to create the ML models that are the ultimate goal of this project.

3.2 Predictive Task and Modelling Approaches

The ultimate aim we want to achieve with this research is to automatically predict rotation periods of stars from the *Kepler* catalogue, using a “standard” set of structured predictors, by resorting on ML methods.

3.2.1 Problem Formulation

Since stellar rotation periods are positive real numbers, that can range from nearly zero up to tens of days (Santos et al., 2019; McQuillan, Mazeh and Aigrain, 2014), we frame the problem as a regression task. Therefore, our problem can be enclosed by eqs. (2.1) and (2.2), where the target variable y and the p predictors $\mathbf{x} = (x_1, \ldots, x_p)$ are respectively the rotation period, $p_{\rm rot}$, and the set of variables described in section 3.1 and table A.1:

$$p_{\rm rot} = f(x_1, x_2, \ldots, x_p) + e. \tag{3.1}$$

In this equation, p is an integer varying up to 180, according to the number of variables making up the data set used to train the model (DS0 to DS10). The model, to be developed by ML methods, can be written as

$$\widehat{p_{\rm rot}} = \hat{f}(x_1, x_2, \ldots, x_p), \tag{3.2}$$

where $\hat{f}$ is the estimation for f. As already pointed out in section 2.1, we will not be concerned with the exact form of $\hat{f}$, but rather with the accuracy of the predictions yielded by the models.

An important condition for us was that the models were simultaneously robust, had good predictive performance, and were computationally cheap. Given the tabular nature of the data available and from what was explained in section 2.2, we decided to use tree-based ensembles methods, in particular RFs and XGBoost, to tackle our research problems. The former, because they are reliable, easy to use, and have good predictive performance (Torgo, 2011); the latter, because although harder to tune than RFs, they are fast and have recently proven to yield the best results for predictive tasks out of structured data (Géron, 2017; Raschka and Mirjalili, 2017), and we wanted to check how much we could improve results obtained with RFs by using a GB approach.

We point out that, in both methods, the algorithms contain several hyperparameters (recall sections 3.2.2 and 3.2.3), which are not contemplated in eq. (3.2). As we saw in section 2.1.2, these parameters are used to tune and improve the performance of the models. Optimising them using a resampling technique is crucial and the best way to build a robust model with good predictive performance. In section 3.3.1, we will describe the strategy we have followed to optimise those hyperparameters.

The proposed methods aim at improving the predictive performance of existent models, such as the RF classifier published by Breton et al. (2021), so that rotation periods can be estimated for thousands of K and M stars from the *Kepler* catalogue, effortlessly and in a timely manner. Ultimately, we expect that our results will be useful in predicting rotation periods for stars of other spectral classes and from other catalogues, and that our approach will serve as a stepping stone for future results.

Interpretability of the models. A drawback of RFs and XGBoost is that they are considered black-box algorithms, in the sense that gaining insight of $\hat{f}$ in eq. (3.1) will be hard, due to the large number of trees that will compose the ensembles. We will be relying on the calculation of

the variables' importance, which is provided by the implementations of both algorithms in R, to extract interpretable information about the contribution of the different variables.

3.2.2 The Random Forest Approach

As we have seen in section 2.2.7, RFs are widely known for their good scalability and ease of use. A RF is an ensemble of DTs: the algorithm grows unpruned trees concurrently and combines the individual predictions, in order to get a final outcome. The main idea behind a RF is to build a robust model out of multiple (deep) DTs, each of which exhibiting high variance, but that when combined generate a model which is less susceptible to overfitting and is able to increase the overall generalisation performance (Raschka and Mirjalili, 2017). The basic ideas to build an RF algorithm are outlined in fig. 3.8, and can be described in the following steps:

1. Generate a bootstrap sample of size n from the training set, *i.e.*, randomly choose n observations with replacement, where n is total number of cases in the set.
2. Grow a DT out of the bootstrapped sample; at each node:
   2.1. Randomly pick m_{try} predictors, without replacement;
   2.2. Split the node using only one of those m_{try} predictors, *i.e.*, split the subset of examples using the feature that provides the best result according to the objective function* — for instance, minimising the entropy or the MSE.
3. Repeat the steps 1 and 2 n_{tree} times.
4. Aggregate the prediction by each tree in the following way:
   - **classification task**: assign the class label by majority of vote.
   - **regression task**: assign the response value to the average of the predictions of all DTs.

START;
Select the number of models to build, n_{tree};
for $i = 1$ *to* n_{tree} **do**
  Draw a bootstrap sample of size n of the training data;
  Train a tree model on this sample;
  for *each tree node* **do**
    Randomly select $m_{\text{try}} < p$ of original predictors;
    Select best predictor among the m_{try} features and partition data;
  end
  Use typical tree model stopping criteria to determine when tree is complete;
  Do not prune tree;
end
Aggregate predictions by individual trees into class label or average value depending if classification or regression task, respectively;
END;

Figure 3.8: Basic ideas to build an RF algorithm. The p parameter denotes the total number of features in the data set (adapted from Kuhn, Johnson et al., 2013).

* In the context of ML, the objective function refers to a quantity to be maximised or minimised in a specific optimisation problem (Hastie, Tibshirani and J. Friedman, 2009; Kuhn, Johnson et al., 2013).

Since DTs are the fundamental components of RF, they can handle many types of predictors without requiring pre-processing. The only typical exception are missing values which, in most implementations of the algorithm, are not accounted for and need to be handled with some particular strategy (Torgo, 2011). A common approach is *imputation*, in which missing values are replaced by some value obtained with a statistical measure, such as the mean or the median, calculated on the predictor containing the missing values.

Random Forests are not as interpretable as DTs; however, they are robust to noise from the individual trees, requiring no pruning and no major tuning of the hyperparameters. In practice, the only hyperparameter that requires attention during optimisation is n_{tree}, the number of trees composing the forest. Similarly to bagging, RFs do not overfit when the number of bootstrapped training sets is increased. The larger this value, the better the performance of the random forest, at the expense of computational cost (Raschka and Mirjalili, 2017). Nonetheless, since RFs, similarly to bagging, are ensembles of *independent models*,* they are prone to be parallelised, which can accelerate considerably the training phase (Torgo, 2011). Therefore, in practice a sufficiently large number of trees is used in order to decrease the error rate (James et al., 2013).

Hyperparameters in random forests. The most common hyperparameters of a RF are (*i*) the number of trees in the forest, n_{tree}, (*ii*) the fraction of the original data set that is assigned to each tree (related to the size n of the bootstrap sample), (*iii*) the minimum number of observations required to split a node, (*iv*) the maximum depth of a tree, and (*v*) the number of variables m_{try} to consider when looking for the best split (Breiman, 2001). Of those, the most important, that is, the ones that have the biggest impact in the performance of the model, are arguably n_{tree}, n, and m_{try}.

The number of trees (n_{tree}, `num.trees`),† has a default value of 500 in the **ranger** package. Typically, the performance of the model is improved by increasing n_{tree}, but the learning process becomes slower, which can be compensated by a parallel computation strategy.

The sample of observations assigned to each tree, `sample.fraction`, has a default value of 1 for sampling with replacement (bootstrapping, the default in **ranger**), and 0.632 for sampling without replacement. The bias-variance trade-off of the RF is controlled via this parameter. Taking `sample.fraction` equal to 1 usually provides a good bias-variance trade-off. Smaller fractions lead to greater diversity, since it decreases the probability that a particular observation is included in the bootstrap sample; thus, the *randomness* of the forest is increased, and less correlated trees are created, which can help to reduce the effect of overfitting. While smaller gaps between the training and testing performances can be observed, reducing the bootstrap sample size typically decreases the overall testing performance (Raschka and Mirjalili, 2017).

The minimum number of cases necessary at a leaf node, `min.node.size`, defines the minimum size of terminal nodes.‡ It implicitly sets the depth of the trees in a random forest. Smaller trees are grown when this parameter is large, making the training process faster; on the contrary, when `min.node.size` is small, trees become deeper and prone to overfitting, and the learning process

* In ensembles of independent models, each individual learner is obtained independently of the others.

† Words in `typewriter` font indicate parameters as they are named in R packages.

‡ Nodes with size smaller than `min.node.size` can occur (Wright, Wager and Probst, 2019).

takes more time to complete. The default values are different for classification and regression. In the **ranger** package, the default `min.node.size` is 5 for regression and 1 for classification.

The maximum depth of the trees in the forest, `max.depth`, has a default value of `NULL` or 0 in **ranger**, corresponding to trees with unlimited depth; when set to 1, meaning that each tree is split once, the forest becomes composed of decision stumps.

The number of variables to possibly split at in each node (m_{try}, `mtry`), determines the diversity of individual trees. For classification, its default value is the rounded down square root of the number variables, $\lfloor\sqrt{p}\rfloor$, and for regression the rounded down third of the number of features, $\lfloor p/3\rfloor$. Hence, at each node, the algorithm uses a subset of the available predictors to select the best split for each tree of the random forest. This hyperparameter affects directly the model's performance, because large values of m_{try} decrease the diversity of individual trees and increases the training time. When there is a large number of correlated features in the data set, choosing a small value of m_{try} will typically be helpful in building a random forest (Hastie, Tibshirani and J. Friedman, 2009).

Variable importance in random forests. As seen in section 2.2.6, while growing the individual trees, the split criteria, that is, the (locally) optimal condition the algorithm applies is decided by the reduction of some measure, known as the decrease of the *node impurity*. For classification, the impurity is commonly measured by either the *Gini index* or the *entropy*. For regression trees, the impurity is typically measured with the estimated response variance (Wright and Ziegler, 2017).

When a tree is trained, the algorithm accounts for how much each variable decreases the weighted impurity of the model. In the case of a forest, the decrease in impurity from each of the features is averaged out, and the predictors are ranked in terms of importance according to this measure. The impurity based ranking of variables has some drawbacks. On the one hand, when categorical variables are present in the data set, feature selection based on impurity is biased towards variables with more categories (Strobl et al., 2007); on the other hand, in the presence of correlated features, each will be likely selected as a predictor, and once the model picks one, the importance of the others within the subset of correlated variables is reduced, since the impurity they can decrease has already been reduced by the first selected feature. Consequently, the other features will have a lower reported importance. This can lead to the wrong conclusion that one of the features is a strong predictor, while the other correlated variables are unimportant, thus affecting the interpretability of the data. This effect is attenuated by the random selection of features at each node, but it is generally not completely removed.

In the next section, we will address the GB technique in more detail than it was done in section 2.2.7.

3.2.3 The Extreme Gradient Boosting Approach

In section 2.2.7 we briefly touched on boosting methods. Here, we are going to look at two of them—*AdaBoost* and *gradient boosting*—in more detail.

Boosting can combine (or *boost*) several weak learners, *i.e.*, models that predict marginally better than random, to produce a strong learner, an ensemble with a superior generalised error rate. Boosting algorithms are known as ensembles of *coordinated models*, because each of its members depends on the others (Torgo, 2011).

Several boosting algorithms are available, most of which train predictors sequentially, so that at each iteration a weak learner is added, trying to correct its predecessor (Géron, 2017). Weak models are added to the ensemble with weights that reflect the ensemble's performance. In each iteration, the weight of those observations that are poorly predicted by the current ensemble is increased.

Almost any ML method with hyperparameters can be made into a weak learner. Decision trees make excellent weak learners for boosting, because (*i*) they can be made flexible by restricting their weight; (*ii*) individual trees can be added together to generate a prediction, much like individual predictors in a regression model; and (*iii*) they can be generated very quickly.

The most popular boosting methods are *adaptive boosting* (AB), or *AdaBoost*, and *gradient boosting* (GB).

AdaBoost (Freund and Schapire, 1997) is called "adaptive" in the sense that at each iteration, when building a new weak learner (*e.g.*, a DT), the weights of the training cases are adjusted: the weights of those instances that were wrongly predicted are increased, so that the new learner will focus on them, trying to accurately predict them. The algorithm can be applied to both classification and regression tasks. Figure 3.9 illustrates the basic concept underpinning AdaBoost. Represented is a training data set for binary classification.

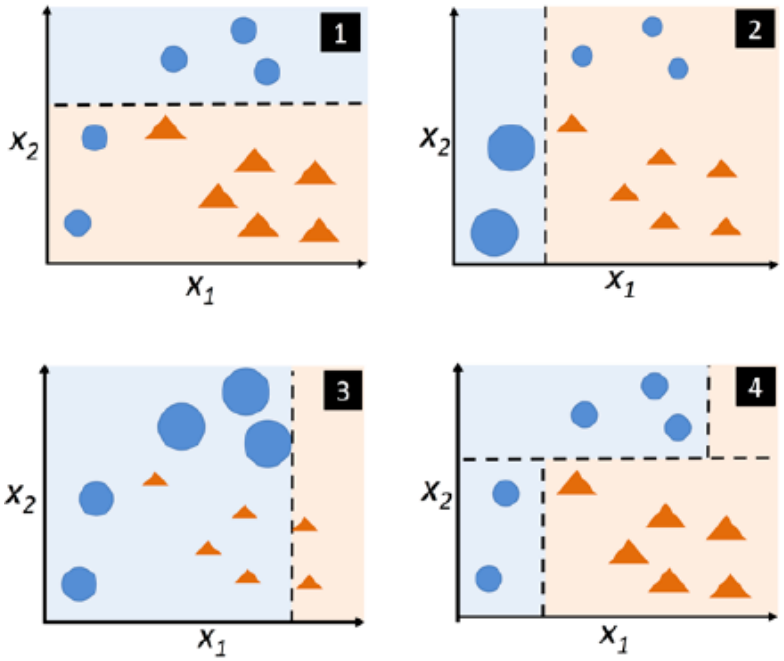

Figure 3.9: Basic concept behind AdaBoost. Source: Raschka and Mirjalili (2017).

In sub-figure 1, equal weights are assigned to the examples. A decision stump (horizontal dashed line) is trained to classify the instances of the two classes (circles and triangles) and to

possibly minimise the cost function (or the impurity, in the case of decision tree ensembles). Sub-figure 2 represents the next iteration, in which a larger weight is assigned to the two blue circles which were misclassified in the previous step, and the weight of the correctly classified observations is lowered. The new decision stump focus on the instances that have the largest weights—the examples that supposedly are harder to classify. In sub-figure 3, the three blue circles that were misclassified in the previous step are assigned a larger weight. Assuming that this AdaBoost ensemble consists of only three rounds of boosting, the three weak learners previously trained on re-weighted subsets are combined by a weighted majority vote, as shown in sub-figure 4 (Raschka and Mirjalili, 2017). At the end, AdaBoost produces an additive model, that can be mathematically represented as (Torgo, 2011)

$$H(x_i) = \sum_k w_k h_k(x_i), \tag{3.3}$$

where w_k is the weight of the weak base model $h_k(x_i)$. In the first iteration, the algorithm assigns the same weight to all training instances, $d_1(x_i) = 1/n_t$, where n_t is the sample size. At iteration r, the algorithm builds the weak model $h_r(x_i)$ so that this model minimises the weighted training error, given by $e_r = \sum_i d_r(x_i) I(h_i \neq h_r(x_i))$, where $d_r(x_i)$ is the weight of the instance (x_i, y_i). The weight of the weak base model $h_r(x_i)$ is obtained with

$$w_r = \frac{1}{2} \ln \left(\frac{1 - e_r}{e_r} \right). \tag{3.4}$$

The iteration $r+1$ receives the same data sample, but with the weights of the instances changed to reflect the unsuccessful predictions of the current ensemble of models:

$$d_{r+1}(x_i) = d_r(x_i) \frac{\mathrm{e}^{-w_r I(y_i \neq h_r(x_i))}}{z_r}, \tag{3.5}$$

where z_r is a normalisation factor chosen to make all d_{r+1} sum up to 1 (Torgo, 2011; Freund and Schapire, 1997). The general steps of an AdaBoost algorithm for two-class problems is highlighted in fig. 3.10.

Gradient boosting (J. H. Friedman, 2001) is a method able to address classification and regression tasks, based on the idea of *steepest-descent minimisation*. Given a loss function L and a weak learner (*e.g.*, regression trees), it builds an additive model as in eq. (3.3) that tries to minimise L. The algorithm is typically initialised with the best guess of the response, such as its mean in the case of regression, and it tries to optimise the learning process by adding new weak learners that focus on the residuals (errors) of the current ensemble. The model is trained with a set formed by the cases (x_i, r_i), where

$$r_i = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}, \tag{3.6}$$

and L is a loss function. Hence, the residuals yield the gradients which, for the most common loss functions, are easy to calculate. For example, for the squared loss function $\frac{1}{2}[y_i - f(x_i)]^2$, the gradient is equal to $[y_i - f(x_i)]$ (Torgo, 2011). The current model is added to the previous one,

START;
Let one class be represented by $+1$ and the other class with the value -1;
Let each example have the same starting weight ($1/n$);
for $k = 1$ *to* K **do**
  Fit weak learner using the weighted instances and compute the k^{th} model's misclassification error, err_k;
  Compute the k^{th} stage value as $\ln[(1 - err_k)/err_k]$;
  Update sample weights giving more weight to incorrectly predicted samples and less weight to correctly predicted samples;
end
Compute boosted learner's prediction for each sample by multiplying the k^{th} stage value by the k^{th} model prediction and adding these quantities across k;
if $sum > 0$ **then**
  label of instance $\leftarrow +1$;
else
  label of instance $\leftarrow -1$;
end
END;

Figure 3.10: General ideas to build an AdaBoost algorithm for a two-class problem (adapted from Kuhn, Johnson et al., 2013).

and the process continues until a stopping-condition is met (*e.g.*, the user-specified number of iterations).

The general steps for a regression algorithm for GB can be found in fig. 3.11. In this example, regression DTs were used as the base learners, in which case GB has two tuning parameters—*tree depth* and *number of iterations*—and the squared error was used as the loss function.

START;
Select the tree depth, D, and the number of iterations, n_{tree};
Compute the average of the response, $\bar{y}$;
Use $\bar{y}$ as initial predicted value for each sample;
for $k = 1$ *to* n_{tree} **do**
  Compute the residual (difference between the observed value and the *current* predicted value), for each example;
  Fit a regression tree of depth D, using the residuals as the response;
  Predict each sample using the regression tree fit in the previous step;
  Update the predicted value of each sample by adding the predicted value from the previous iteration to the predicted value generated in the previous step;
end
END;

Figure 3.11: General steps for a simple GB algorithm for regression. Adapted from (Kuhn, Johnson et al., 2013).

In boosting, the variable importance is a function of the reduction in the squared error (Freund and Schapire, 1997; J. H. Friedman, 2001; Kuhn, Johnson et al., 2013). Typically, the importance profile for boosting has a stepper slope than the one for RFs, because in boosting DTs are dependent on each other, and hence will have correlated structures as the algorithm follows the gradient.

Several of the same features will thus be selected across the trees, increasing their contribution to the importance metric (Kuhn, Johnson et al., 2013). It is natural that differences exist in the importance profiles between RFs and boosting, both in terms of order and magnitude of importance. The two profiles can be seen as different perspectives of the data, and each should be used to provide insights of the relationships between the predictors and the target variable.

As we have seen in section 2.2.7, a successful implementation of GB is XGBoost, which was designed with speed, efficiency of computer resources, and model performance in mind. The algorithm is suited for structured data. It is able to automatically handle missing values, it supports parallelisation of the tree building process, it automatically takes care of early stopping when the training performance is not evolving after some predetermined iterations, and it allows to resume the training of a model by boosting an already fitted model on new data (T. Chen and Guestrin, 2016).

The method implements the GB algorithm, with minor improvements in the objective function. A prediction of the response is obtained from a tree-ensemble model built upon K additive functions:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \tag{3.7}$$

where the function f_k belongs to the functional space $\mathscr{F}$ of all possible DTs. Each f_k corresponds to an independent tree learner. The final prediction for an example is given by the aggregation of the predictions of the individual trees. The set of functions in the ensemble is learnt by minimising a regularised objective function, $\mathscr{L}(y_i, \hat{y}_i)$, consisting of two parts:

$$\mathscr{L}(y_i, \hat{y}_i) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \omega(f_k). \tag{3.8}$$

The first term of the right hand side of eq. (3.8) is the *training loss*, $L(y_i, \hat{y}_i)$, and the second term is the *regularisation term*. Here, ℓ is a differentiable convex loss function measuring the difference between the predicted and the true values of the response. The training loss measures how predictive the model is with respect to the training data. A common choice for L is the MSE. The regularisation term controls (penalises) the complexity of the model. It helps to smooth the final learn weights, to avoid overfitting. In practice, the regularised objective function tends to select models employing simple and predictive functions (T. Chen and Guestrin, 2016). A model is learnt by optimising the objective function above.

Hyperparameters in XGBoost. XGBoost is suitable for hyperparameter optimisation in a timely manner. Arguably, the most commonly tuned XGBoost hyperparameters are (*i*) the number of sub-trees to train, (*ii*) the learning or shrinkage rate, (*iii*) the number of variables to grow in each node or tree, (*iv*) the maximum tree depth, (*v*) the subsampling fraction, (*vi*) the minimum child weight, and (*vii*) the reduction or complexity cost.

The number of sub-trees to train, `nrounds`, is used as a stopping criteria for the algorithm, preventing overfitting and the ability of the model to memorise the training data.

The learning rate (η, `eta`) is a weighting factor for the corrections by new trees when they are added to the model. It is the step size shrinkage used in each boosting step to make the process more conservative and, hence, to prevent overfitting. It controls how much information from each new tree is used in boosting. The learning rate is a real value between 0 and 1. When η is close to zero, a small portion of information from each tree is used for boosting; when set to 1, all information available from a new tree is used. Setting η to values less than 1 introduces less corrections for each tree added to the model, resulting in more trees that must be added to it. Smaller learning rates turn the computation slower and require more rounds (and, hence, more trees) to achieve the same reduction in the residuals as larger values of η, but they optimise the chances of reaching the best model. Larger learning rates result in faster computations, but can cause the model to miss the optimum completely. Therefore, the learning rate must be set as low as possible, which most of the times is dependent on the complexity of the data and on the available computer resources.

The number of variables re-sampled in each new node or tree, a process which occurs once for every tree which is built, is helpful in controlling overfitting. While using all the features available in a new node/tree makes the algorithm to converge faster, using a fraction of the predictors may result in more robust models.

The maximum tree depth is used to control the size of the trees. Deeper trees have more terminal nodes, fit more data, and make convergence faster, in the sense that less trees are needed. This hyperparameter is typically defaulted to 3. The greater the value, the deeper the tree is, with 0 indicating a tree with no depth limit. A deeper tree might increase the performance, but also the complexity and chances to overfit; moreover, deeper trees imply that more information is used from the first trees than from the final trees of the algorithm, making the latter less important on the loss function. Boosting benefits from using information from many trees, and so too large trees are not desirable. Taking this parameter much smaller than 3 can significantly reduce the number of expected useful interactions. Using fractions of whole numbers for the maximum tree depth usually leads to over-tuning, thus increasing the training time and not yielding noticeable benefits in terms of overall performance. A rule of thumb consists in increasing the maximum tree depth by unit steps and seldom above 5—the developers of XGBoost have pointed out for the fact that the algorithm "aggressively consumes" memory when training deep trees.* Therefore, it is important to set the maximum tree depth according to the available memory.

The subsample parameter determines whether a regular or a stochastic boosting is being determined. In the latter, the base learners are fit using subsamples drawn at random, without replacement, from the training data set. Values greater than 0 and less than 1 correspond to the stochastic case, while regular boosting is obtained with subsample set to 1. Therefore, this parameter represents the fraction of observations to be sampled for each tree that is constructed, occurring once in every boosting iteration. Stochastic boosting is useful for limiting the influence of outliers on the final model, since they are not considered in several subsamples. Lower values prevent overfitting, but might lead to underfitting.

* https://github.com/mljar/mljar-supervised/issues/13

The minimum child weight controls the minimum number of observations in a terminal node, by setting a lower bound on it. The building process does not proceed to further partitioning if the step results in a leaf node with the sum of instance weight (hessian) less than `min_child_weight`. In a regression task, this simply corresponds to the minimum number of instances needed to be in each node. This parameter can be as low as 1, in which case trees are allowed to have terminal nodes with only one observation. Larger values imply a stronger regularisation, in the sense that more instances are required in terminal nodes, making each tree smaller and, consequently, the algorithm more conservative — this limits a possible perfect fit on some observations, but the model will be less prone to overfitting.

The reduction or complexity cost (γ, `gamma`) is a Lagrangian multiplier, a pseudo-regularisation parameter, depending on other parameters, which controls the minimum reduction in the loss function required to grow a new node in a tree. It can vary from 0 to infinity, and it is dependent upon both the data set and the other hyperparameters. Therefore, one data set can have multiple ideal values for γ, depending upon how the other hyperparameters are set. The γ parameter leads to shallower trees or, at least, trees with fewer leaves, by restricting the splitting process. It is sensitive to the scale of the loss function, which in turn is dictated by the scale of the response variable. According to the XGBoost documentation, the loss function implemented in the package is defined as

$$L(\theta) = \sum_i \ell(\theta), \tag{3.9}$$

where $\ell(\theta)$ can assume several forms. For example, for the quadratic loss, $\ell(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$, where y_i and $\hat{y}_i$ correspond respectively to the true and predicted values (T. Chen and Guestrin, 2016). Typically, for an RMSE reduction of at least m in a split, γ must be of the order of $(m \times s)^2$, where s is the subsample size.* Taking γ different from 0 helps the algorithm to stop from growing unnecessary trees, that could in principle barely reduce the in-sample error and could potentially result in overfitting. The larger γ, the more conservative the algorithm will be, that is, the stronger the regularisation and, hence, the less prone to overfitting the model will be. When it is set to 0 (the default), no regularisation is applied to the model. Contrarily to the maximum depth and the minimum child weight, which regularise using "within tree" information, γ performs regularisation using "across trees" information. This hyperparameter becomes relevant when one intends to use shallower trees to combat overfitting (Laurae, 2016).

Although XGBoost has more hyperparameters available for tuning, these are the ones considered affecting the most the bias/variance decomposition of the MSE, *i.e.*, the level of underfitting and overfitting, thus having the biggest impact on the performance of the model.

In the following section, we will describe the experimental design and how we tuned the hyperparameters of the methods we used to create the ML models.

* https://www.r-bloggers.com/2018/07/tuning-xgboost-in-r-part-ii/

3.3 Experimental Design

Two ensemble methods were used to build regressor models: *random forests* (RF), and *gradient boosting* (GB). Each model was trained using the hold-out method, where most of the data sets described in section 3.1 were split into training and testing subsets, in a 75-to-25 percent ratio. The exceptions were DS9 and DS10, for which we opted to do an 80-to-20 percent ratio split, due to the reduction in the number of observations. During the training phase, we used a 10-fold cross-validation (10CV) with five repetitions, where the best values for the hyperparameters were sought using a searching grid.*

In all cases, the testing sets remained unseen by the models until the prediction and model assessment phase. That is, the instances belonging to the testing sets were no part of the model learning process in any time, behaving as new, unknown observations, as it would happen in a real-life scenario.

3.3.1 Hyperparameter Values Optimisation

In face of what was exposed in sections 2.1.2, 3.2.2 and 3.2.3, we designed two experiments, one for RFs and other for XGBoost, in which we performed grid search of the best values for the hyperparameters of each method. The hyperparameters for the models were chosen using 10CV with five repetitions. In the following, we will outline how the tuning grids were built, and will justify our choices for the initial set of values for those parameters.

Hyperparameter tuning in random forests. The RFs models were trained and created using the **ranger** R package. We performed optimisation of the following parameters:

- `min.node.size`: the minimal node size, *i.e.*, the minimal number of instances at each node (larger numbers correspond to less overfitting, that is, to more pruned trees); it was varied between 2 and 10, by unit steps.
- `mtry`: the number of randomly sampled predictors to possibly split at in each node; it varied in unit steps within the interval $\lfloor n_{\text{preds}}/3 \rfloor \pm 3$, where n_{preds} is the number of predictors.
- `splitrule`: the splitting rule; it was chosen between two possibilities: `variance` and `extratrees`, where the number of random splits to consider for each candidate splitting variable was kept at 1 (the default).

Each forest was composed of 1000 trees, with a fraction of 70 % of all the observations for each tree. Although we would, in principle, have benefited from varying the number of trees, access to computational resources was always very limited and we chose to keep this value unchanged. The variable importance was assessed by means of the impurity measure, which corresponds to the variance of the response variable.

* We have also tried to use random search in looking for those optimal values, in an attempt to reduce the computational executing time, but we were never able to outperform the results obtained with an exhaustive search. Hence, since it was possible to train models using the largest data set in a timely manner, we decided to use a thorough grid search in all scenarios.

Hyperparameter tuning in extreme gradient boosting. The GB was implemented via its optimised XGBoost framework. We looked for the best values of the parameters defined in section 2.1.2 via a two-step grid search: firstly, the best value was sought within a set of possible values; then, the search was refined on the learning rate and on the subsample ratio of training instances, by varying the parameter in a finer grid centred on the best value found in the previous step. No more than 1000 iterations were carried out for each submodel, since we activated a stopping criterion, in which the learning would end for any particular model if the performance was not improving for five rounds. The values of the grids for each parameter were the following:

- `colsample_bytree`: the fraction of columns to be randomly subsampled when constructing each tree; it was sought within the set $\{1/3, 1/2, 2/3\}$, also during the first grid search.*
- `eta`, the *learning rate*, was sought within the values $\{0.05, 0.1, 0.2, 0.3\}$ in a first stage; then, the search was refined by looking for five values centred on the best η found in the first iteration, each 0.01 apart.
- `gamma`: the minimum loss reduction required to further partition a leaf node of the tree; it was kept at its default value of 0 in most of the models; however, an optimal value was sought on the final models, built on top of the selected subset of the most important features in the previous models and with a sub-selection of observations including only stars with rotation periods no grater than 45 d; it was then chosen within the set of values $\{1, 10, 30, 100\}$.
- `max_depth`: the maximum depth per tree; the best value was chosen on the set of integer values $\{3\mathtt{L}, 5\mathtt{L}\}$, during the first iteration.
- `min_child_weight`: the minimum sum of instance weight (a Hessian matrix)† needed in a child node; similarly to the γ above, this parameter was kept at its default value of 1 in most of the scenarios, and an optimal value was only sought while learning the final models, in which case the values for the grids were 1L, 10L, and 50L.
- `subsample`: the subsample ratio of the training instances; we opted for a stochastic boosting, and so during the first grid search, `subsample` was chosen within three possible values: 0.25, 0.5, and 0.75; subsequently, the best value of the parameter found during the first grid search iteration was refined, by looking for the best of five values centred on that best value, each 0.05 apart.

* Two more hyperparameters belonging to this family of subsampling columns were initially optimised: `colsample_bylevel` and `colsample_bynode`. However, no improvement in the model's performance was observed when including them, and since they work cumulatively, at the end only `colsample_bytree` was kept during the optimisation process.

† If $f(x, y)$ is a differentiable function with continuous second derivatives on a neighbourhood of a critical point $(x, y) = (a, b)$, the *Hessian* or *Hessian matrix* of f at (a, b) is the symmetric matrix of second derivatives

$$H_{(a,b)} = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(a,b) & \frac{\partial^2 f}{\partial x \partial y}(a,b) \\ \frac{\partial^2 f}{\partial y \partial x}(a,b) & \frac{\partial^2 f}{\partial y^2}(a,b) \end{pmatrix}.$$

We say that (a, b) is a *critical point* when $\partial f/\partial x(a, b) = \partial f/\partial y(a, b) = 0$ (Callahan, 2010).

3.3.2 Final Models

After training and assessing the performance of the models described in section 3.3, data were narrowed down to a subset of the most relevant cases and most important predictors, as explained in section 3.1.1 (page 41), on top of which five additional models were trained.

Three models—two based on RFs and another on XGBoost—were learnt from the DS9 data set, in which stars with rotation periods greater than 45 d and shorter than 7 d were removed, and only features with the greatest importance in the models built on top of DS2, DS3, DS4, DS6, DS7, and DS8 were included. By doing so, on the one hand, we were expecting to increase the predictive performance of the models, since (*i*) stars with $P_{\rm rot} < 7$ d may be close-in binary stellar systems or classical pulsator star candidates (Breton et al., 2021), and (*ii*) due to the 90-day quarters of Kepler, rotation periods above 45 d are not reliable; on the other hand, by dropping the least important features, as identified during the learning of the RFs and XGBoosts models using DS2 to DS8 data sets, we were hoping for a significant reduction of the training process, without affecting the predictive performance of the models. In the RF models, the grid searches were similar to the ones presented in section 3.3.1, with exception for the parameter `splitrule`, which remained fixed and equal to "variance"; such grids gave rise to 63 models. For the XGBoost model, the searching grid was refined in the very first iteration, by increasing the set of possible values for `eta`, `max_depth`, and `subsample`, and by including the `gamma` and `min_child_weight` parameters:

- `colsample_bytree`: $\{1/3, 1/2, 2/3\}$ (unchanged).
- `eta`: varied between 0.03 and 0.1, by 0.01 steps.
- `gamma`: $\{1, 10, 30, 100\}$.
- `max_depth`: $\{5\mathtt{L}, 6\mathtt{L}, 7\mathtt{L}\}$.
- `min_child_weight`: $\{1\mathtt{L}, 10\mathtt{L}, 50\mathtt{L}\}$.
- `subsample`: varied between 0.60 and 1.0, by 0.05 steps.

Not counting with the 10CV with five repetitions, this grid gave rise to 7776 models during training.

Finally, three new models were created out of DS10—one based on RFs and two on XGBoost—where only the ten most important features identified in the models learnt from DS9 were included. By doing so, we wanted (*a*) to assess how much the predictive performance of these models would decrease, and (*b*) to test if it is possible to deploy models easy to train and that could yield reliable stellar rotation periods. We kept the same grid search for the RF model (63 submodels during training), but given the reduced number of features in DS10, we increased the set of available values for the XGBoost hyperparameters as follows:

- `colsample_bytree`: $\{1/3, 1/2, 2/3, 3/4\}$.
- `eta`: varied between 0.03 and 0.1, by 0.01 steps.
- `gamma`: $\{1, 10, 30, 100\}$.
- `max_depth`: $\{3\mathtt{L}, 4\mathtt{L}, 5\mathtt{L}, 6\mathtt{L}, 7\mathtt{L}\}$.
- `min_child_weight`: $\{1\mathtt{L}, 10\mathtt{L}, 30\mathtt{L}, 50\mathtt{L}\}$.
- `subsample`: varied between 0.50 and 1.0, by 0.05 steps.

This grid gave rise to 28 160 submodels during training, without counting with the 10CV with five repetitions.

The training process of all the final models, obtained with data sets DS0 to DS10, and the corresponding results are described in detail in chapter 4.

3.3.3 Performance Assessment Methodology

After the training phase, predictions were produced on each of the testing sets, and the resulting rotation periods were compared to the reference values contained in the S19 catalogue.

We used five metrics to assess the predictive quality of the models produced in this thesis: (*i*) the mean absolute value of the relative residuals, μ_{err}, (*ii*) the RMSE, (*iii*) the MAE, (*iv*) the interval-based accuracy, acc_x, and (*v*) the adjusted coefficient of determination, R^2_{adj}. The index x, which can take the values 5, 10, and 20, corresponds to the percentage width of the zeroing-interval centred on the reference value.

In computing the interval accuracies defined in section 2.1.2 (eqs. (2.11) and (2.13) on page 15, respectively), we note that the width of the intervals centred in the reference values increases with the rotation period, while the width of the intervals around the corresponding rotation frequencies does not vary significantly with the frequency, except on the low frequency regime (*conf.* fig. 3.12). Therefore, because the distribution of frequency widths is more uniform than that of period widths, the rotation periods were firstly converted to rotation frequencies, with the aim of reducing the effect of increasing the width of the accuracy intervals with the period, which would affect comparisons and the assessment of the quality of the models.

The mean of the relative absolute values of the residuals was used as a goodness of fit measure, allowing us to have an idea of how much the models were wrong, on average.

The RMSE and MAE were used together to measure the differences between the predicted and reference values of stellar rotation periods. They were particular useful to assess the performance of the models on the training and testing sets and, thus, to better tune them in order to avoid overfitting—large differences in the RMSE and/or MAE between the training and testing sets are usually an indication of overfitting.

With the interval-based accuracy we were able to convert the assessment of a regression problem into the evaluation of a classification result. That, in turn, allowed us to compare the predictive performance of our models with the one trained by Breton et al. (2021). Since the latter assessed their results within a 10 % interval of the reference values, we used the 10 %-accuracy, acc_{10}, as a benchmark.

The adjusted coefficient of determination, R^2_{adj}, was used to fairly compare the models trained with the different data sets (recall section 2.1.2, page 17). It will also allow future work to be compared with ours, even if the number of predictors in the data sets is different.

The quality assessment of all the models is detailed in sections 4.1 to 4.3, and an overview of the results is discussed in section 4.4.

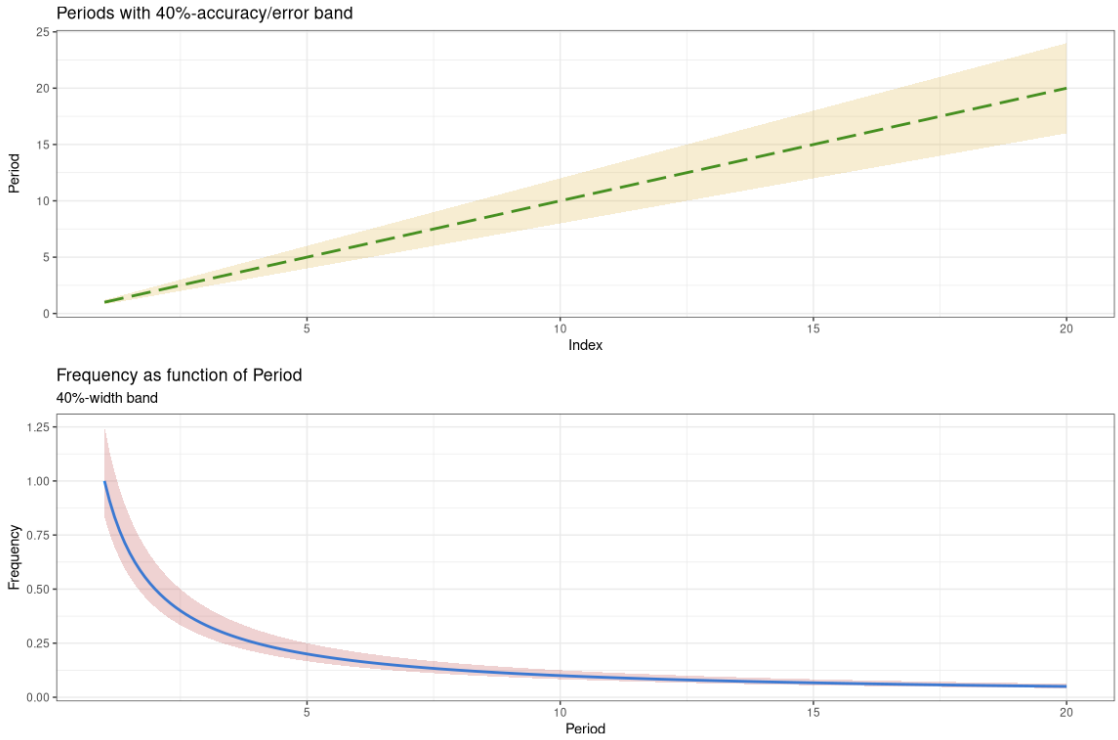

Figure 3.12: Variation of the error bands with the period and the frequency. While the width of the intervals increases linearly with the period (upper panel), it decreases with the frequency, and it does not change significantly for high frequencies (lower panel)—the variation is only relevant in the low frequencies regime. The x-axis in the upper panel represents the position of the periods in the corresponding vector which, coincidentally, are equal to each other.

CHAPTER 4

Prediction of Stellar Rotation Periods

You can use all the quantitative data you can get, but you still have to distrust it and use your own intelligence and judgment.
— Alvin Toffler (1928-2016)

It does not depend on size, or a cow would catch a rabbit.
— Pennsylvania German Proverb

Contents

FOLLOWING the methodology introduced in chapter 3, we addressed the task of predicting the rotation period of thousands of K and M stars from the Kepler catalogue, resorting to ML methods. To accomplish this goal, we applied two ensemble regression techniques to the data sets, *random forests* (RFs) and *gradient boosting* (GB), so as to train models to estimate stellar rotation periods using the sets of features described in section 3.1 and table A.1.

The models were trained by performing a grid search on the selected hyperparameters indicated in section 3.3, with 10-fold cross-validation (10CV) with five repetitions on the training sets. The models were evaluated, and the importance of every feature was assessed with regard to the models' predictive powers using the variance of the responses. The details are presented in sections 4.1 to 4.3 and discussed in section 4.4.

4.1 Results Obtained With Random Forests

Random forests (RFs) are known in the realm of ML to typically yield good results without the need of elaborated hyperparameter tuning (Torgo, 2011). Nevertheless, since the best set of parameter values always depends on the characteristics of the specific problem and data set, we decided to tune three hyperparameters by performing a grid search:

- `mtry` (the number of randomly sampled predictors) was varied in unit steps within the interval $\lfloor n_{\text{preds}}/3 \rfloor \pm 3$, where n_{preds} is the number of predictors;
- `splitrule` (the splitting rule for the nodes) was chosen between the two categories `variance` and `extratrees`; and
- `min.node.size` (the minimal node size) was varied between 2 and 10, by unit steps.

To our knowledge and according to several experiments we conducted during this work, `mtry` and `min.node.size` are arguably the most important hyperparameters when tuning a RF, since they have the most noticeable impact in the outcome of the model. The first of the above parameters is also highlighted by Hastie, Tibshirani and J. Friedman (2009) when building trees and, in particular, RFs. The values of the grid were selected according to **ranger**'s defaults, to wit, $\lfloor n_{\text{preds}}/3 \rfloor$, `variance`, and 5 respectively for `mtry`, `splitrule`, and `min.node.size`. After carrying out the grid search with a 10CV with five repetitions, we found the set of optimal parameters indicated in table 4.1.

Table 4.1: Best set of values for the selected hyperparameters in the case of RF models. The split rule was equal to *variance* in all the models.

Hyperparameter	DS0	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8
`mtry`	65	63	58	40	38	16	14	9	7
`splitrule`	... variance ...								
`min.node.size`	2	4	5	4	3	3	3	3	10

The optimal number of randomly sampled predictors to consider for each tree, `mtry`, naturally decreases with the number of features available for the model but, when normalised by the total number of available predictors in the data set, it increases from approximately 36 % in DS0 to 50 % in DS8.

The optimal minimal number of instances at each node, `min.node.size`, varies between 2 and 5, with exception for the smallest data set (DS8), where it was found to be equal to 10. Hence, all data sets (except DS8) tend to build slightly deeper trees than the ones that would be obtained using the default value of `min.node.size`, and shallower trees are built when using uniquely astrophysical variables. This might be an indication that astrophysical features are weaker predictors of the stellar rotation period than the others, because models need more observations at each node in order to perform a split.

The best split rule was `variance` (the default value in the **ranger** package) for all the models, so there was no need to vary this parameter in future grid searches with these data sets.

The quality assessment of the results obtained by applying RFs to the data to predict stellar rotation periods is compiled in table 4.2. The number of predictors available for the training phase is indicated below the label of each data set. The mean of the relative absolute values of the residuals, $\mu_{\rm err}$, is indicated in the first row. The following three rows report on the interval-accuracies calculated over different widths on the reference values, as described in section 3.3.3. These accuracies were computed in an effort to assess the quality of the models and to compare them to the model developed by Breton et al. (2021). Indicated are also the RMSE and MAE for both the training and testing sets, and the adjusted coefficient of correlation, $R^2_{\rm adj}$ computed on the testing data sets.

From $\mu_{\rm err}$, we can say that, on average, DS0 is roughly wrong 9.7 % of the time (or, equivalently, it is correct approximately 90.3 % of the time), DS1 is wrong circa 15.2 % of the time, and so on and so forth. Typically, all models are wrong on average between 9.7 and 36.6 % of the time. According to this metric, the models created with the data sets DS0 to DS7 can be used, at some extent, for prediction purposes, depending on the error we are willing to accept. With a mean relative predictive error of 36.6 %, DS8 is not suitable for predictive purposes, and we can claim, based on this metric, that classical astrophysical quantities are not enough to train a model that can be used to reliably predict stellar rotation periods.

Table 4.2: Quality assessment of the RF models according to the number of predictors. All quality measures are presented with three significant figures, except the adjusted-R^2 (with four). The top row indicates the number of predictors in each data set. The first row shows the mean absolute value of the relative residuals; the three following rows correspond to the interval-based accuracies defined in section 3.3.3, with widths of 20, 10 and 5 % of the reference values; the last three rows present the root-mean-squared error and the mean-absolute error, both computed on the training and testing sets, and the adjusted coefficient of determination (calculated on the testing set). The best results obtained with the metrics (except for RMSE and MAE) are marked in bold.

Assessment	**DS0** **180**	**DS1** **171**	**DS2** **165**	**DS3** **108**	**DS4** **102**	**DS5** **39**	**DS6** **33**	**DS7** **19**	**DS8** **14**
$\mu_{\rm err}$	**0.0968**	0.152	0.151	0.186	0.128	0.186	0.184	0.208	0.366
$\mathbf{acc_{20}}$ (%)	**92.6**	91.0	91.0	88.8	88.5	63.5	63.5	60.2	29.6
$\mathbf{acc_{10}}$ (%)	**87.7**	86.0	85.9	81.2	82.9	38.7	37.9	36.4	15.2
$\mathbf{acc_{5}}$ (%)	**81.6**	78.7	78.8	71.0	74.5	20.0	19.8	18.4	7.70
RMSE	1.53	1.70	1.70	1.81	1.72	2.20	2.20	2.33	4.51
(train/test)	2.98	3.22	3.23	3.30	3.52	4.22	4.19	4.47	7.08
MAE	0.429	0.500	0.509	0.630	0.544	1.31	1.31	1.36	3.18
(train/test)	0.835	0.997	1.01	1.22	1.16	2.58	2.57	2.72	5.10
$\mathbf{R^2_{adj}}$ **(test)**	**0.9422**	0.9279	0.9293	0.9279	0.9168	0.8828	0.8821	0.8668	0.6463

Overall, the accuracies decrease when the number of available predictors is reduced. Naturally, the accuracy increases when going from 5 % to 20 % widths, because more predictions will lie on

the zero-error interval and count as an event. Considering we are willing to accept errors up to 10 %, we use the 10 %-width accuracy, acc_{10}, as reference. They approximately differ successively from each other between 5 to 8 % in the first five data sets, and between 15 to 25 % in the last four data sets. Thus, the gap becomes larger when the models are trained without the CS and GWPS predictors. This means that in these cases many points lie close to the boundaries of the 10 %-width zeroing interval, which will count as a match as soon as we consider a wider interval for the accuracy.

When using all the predictors, the reference accuracy is close to 88 % (DS0), dropping approximately 1.5 points when the *rotation period* predictors are removed (DS1). Removing the *unknown* Wav variables does not affect the quality of the model, regardless of the number of predictors used for training (from DS1 to DS2, and from DS5 to DS6), and so these features were removed from future model training. Dropping GWPS or CS predictors leads to a 4 points loss in the reference accuracy (DS3 and DS4). The most significant drop in accuracy happens when both GWPS and CS variables are removed (from DS4 to DS5/DS6), in which case the reduction is circa 45 points. When using only the combined *time series* and *astrophysical* predictors, we achieved a value of about 38 % for acc_{10} (DS5 and DS6). There is a drop of 1.5 points to the previous case when using only TS variables (DS7), and a further drop of 21 points when using only Astro predictors (DS8), making this model completely useless for prediction purposes of stellar rotation periods.

The computed accuracy increases considerably when the zeroing intervals are made wider and varied from 5 % to 20 %. In the particular case of DS7, when we are willing to accept an error of 20 % in the predictions, we can attain an accuracy 60 % when measured on the reference values. This value drops to 36 %, if we are willing to accept an error no greater than 10 %.

As expected, the RMSE and the MAE tend to decrease, both on the training and testing sets, as the number of available predictors increases. When measured on the testing set, the former varies typically between 3 and 4.5, and the latter between 0.8 and 2.7—exception to DS8, where RMSE and MAE are above 7 and 5, respectively, indicating that this is the worse of the models. There is some level of overfitting in the models, as the magnitude of the RMSE and the MAE on the training sets is roughly the double on the testing sets.

Instead of computing R^2, the adjusted coefficients of determination were calculated on the testing sets using eq. (2.20), so that this statistical measure would not be affected by the different number of explanatory variables and comparisons between models built from distinct data sets would be possible. It varies between 0.92 and 0.94 for data sets DS0 to DS4, where GWPS and CS variables are used as predictors, approximately between 0.87 and 0.88 in data sets DS5 to DS7, and it is equal to about 0.65 in DS8. When considering the percentage of variability of the response explained by the predictors, the quality of the model is not greatly affected by the number of variables provided TS features are present, as R^2_{adj} remains above 0.87. Only when the model is built uniquely with astrophysical variables its quality is significantly degraded, with the adjusted R^2 dropping to 0.65. This is an indication that traditional astrophysical observables are not good predictors of stellar rotation periods, and further variables (astrophysical, TS, or derived) are needed in order to build a solid ML model. Time series features seem to be a good starting point,

as the model built upon them yielded a goodness of fit (as measured by R^2_{adj}) above 86 %. Without accounting for DS0, which contains rotation periods as explanatory variables, about 92 to 93 % of the variance of the response variable can be explained by the GWPS, CS, TS, and astrophysical features, when combined; circa 87 % of the variance can be explained by the TS variables alone; the remaining 8 % and 13 %, respectively, can be attributed to unknown variables which my be hidden in the time series data and still need to be engineered, or to inherent variability.

At this point, we do not know if these values for the adjusted coefficient of determination accurately reflect the fraction of the response's variation that the models explain, because we would need to compare them to the coefficients of determination obtained in similar studies. Since we are unaware of the existence of any such study, we will be able to compare them only later, when we describe the results obtained with the XGBoost models learnt from these data sets.

The scatter plots of the reference values *vs*. the predicted rotation periods are illustrated in fig. 4.1 and, with more detail, in figs. B.8 and B.9, in the appendix. A blue solid line $y = x$ was added to the charts in order to aid the comparison between the model predictions with the reference outcomes. The red dashed line represents the linear model between the predicted and the reference rotation periods.* The graphs look very similar to each other, showing a good linear relationship between the predicted and real rotation periods. The exception goes for DS8, which exhibits greater dispersion than the others, although it continues to show a correlation between the predicted and reference variables.

We can identify outliers in all the plots, both in the region of smaller and larger rotation periods, but they are not in great number. Although the total number of cases with rotation periods beyond the interval $[5, 50]$ days is not large, this is an indication that it is harder for the models to perform predictions in that range of "extreme" stellar rotation periods. The linear model red dashed lines do not indicate that the models are correctly predicting outliers. Therefore, we can infer that the correlation coefficients of figs. B.8 to B.9 and their adjusted counterparts of table 4.2 are not inflated.

Two pairs of plots are practically identical to each other: DS1-DS2, and DS5-DS6. We recall that the difference between the data sets is that the unknown Wav variables have been removed in DS2 and DS6. Therefore, this is further proof that these variables are not relevant during the training of the models.

Figures B.8 and B.9 also show marginal histograms and density plots, and report on the goodness of fit and significance of the linear relation between the predicted and reference values. Visually, the predicted and reference density plots are not too different in terms of centrality, dispersion, and kurtosis. The R-squared indicated in the figures correspond to the coefficient of determination between the predicted and the reference values. They do not differ significantly from the R^2_{adj} reported in table 4.2, as expected, because the total number of observations n is large. The F statistics present in the plots result from testing the null hypothesis that all of the regression coefficients are equal to zero, that is, the models are equal to the baseline (the intercept only model) and do not have predictive capabilities (Archdeacon, 1994). Overall, the F-tests and

* In the graphs of figs. B.8 and B.9, the red dashed line indicates the identity function, $y = x$.

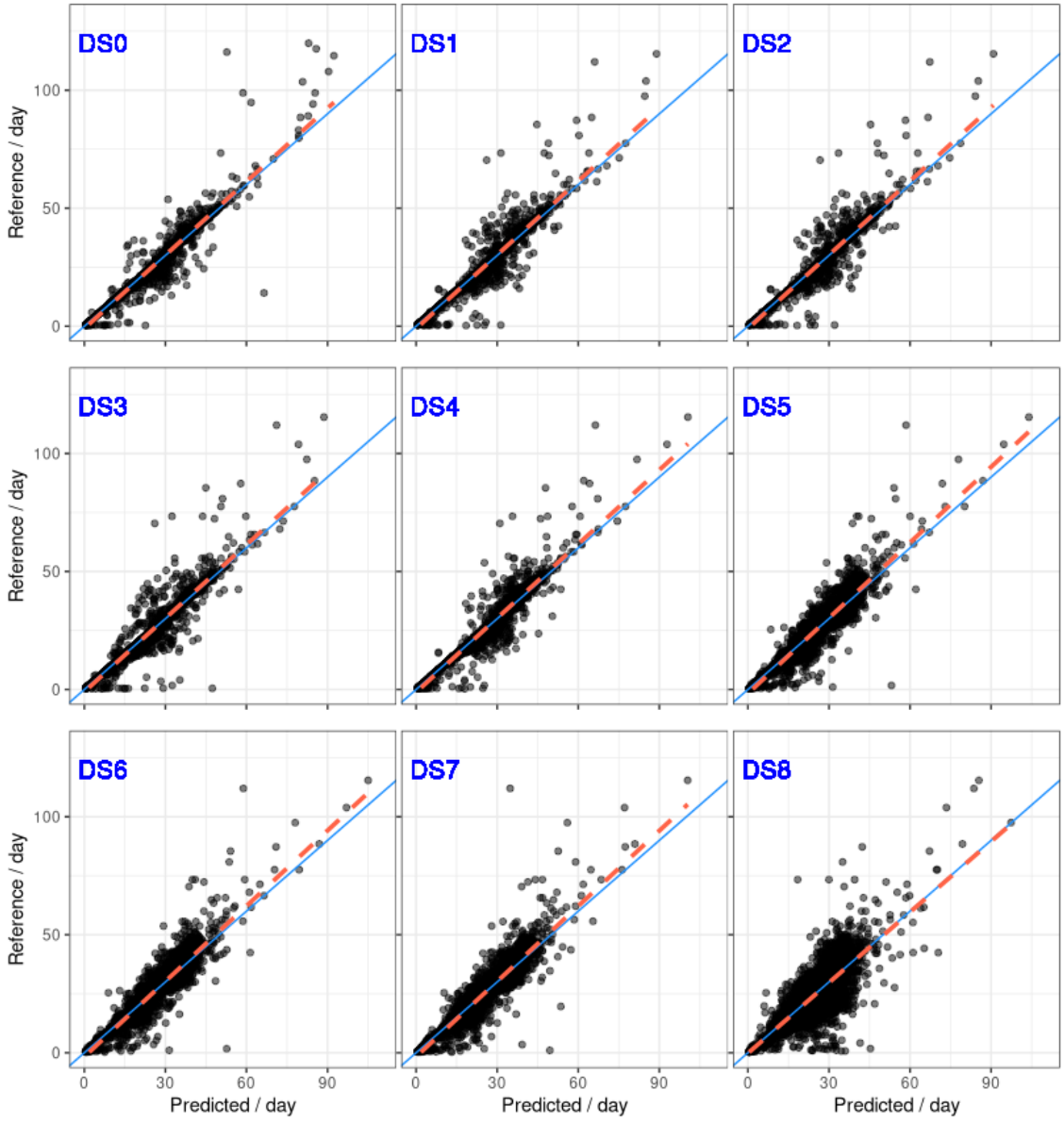

Figure 4.1: Scatter plots of the reference rotation period as a function of the predicted values for RFs models built with the data sets DS0 to DS8. The blue solid lines indicate the identity function, and the red dashed lines represent the linear model between the reference and the predicted values.

the corresponding p-values considered in the figures indicate that the sets of independent variables are jointly significant.

The variables that contribute most to the model naturally depend on the data set but, generally, the rotation periods and ACF, GWPS, and CS predictors dominate the top-10 most important explanatory variables when they are present in the training data set.* The 10 most important variables during each of the nine training processes are indicated in figs. B.1 to B.4.

In the first model, created with the full data set (DS0), the most important variable as measured by the node impurity is the rotation period obtained from the ACF method for the 55-day filter. The rotation period predictors and the central periods for the GWPS and CS methods for the 55- and 20-day filters dominate the top-10 list. The only predictor which is not a rotation period nor is directly related to them is the photometric activity proxy, $S_{\rm ph}$ (fig. B.1).

When training the RF using DS1, $S_{\rm ph}$ remains in the top-10 most important variables, and $T_{\rm eff}$ starts to integrate that list (fig. B.2, top panel). By removing the Wav unknown variables (DS2), there is no change in the quality of the model, as indicated before. The top-10 most important predictors list does not change substantially (fig. B.2, middle panel), and the difference (only on the $10^{\rm th}$ most important variable) can be attributed to the inherent randomness of the training process.

When we remove the CS variables (DS3), the GWPS predictors become the most relevant, together with the FliPer value for $\nu_{\rm C} = 0.7\,\mu$Hz, the photometric activity proxies, and the effective temperature (fig. B.2, bottom panel). Similarly to the DS3 case, in DS4, where we removed the GWPS variables, the CS predictors became the most relevant, followed by the photometric activity proxies, $T_{\rm eff}$, and $F_{0.7}$ (fig. B.3, top panel). In both cases, the central period and the standard deviation of the Gaussians fitted to the GWPS and CS, and the 55-day filter dominate over the variables referring to the amplitude of the Gaussians and to the other filters.

When we remove both the CS and GWPS predictors (DS5), there is a significant drop in the accuracies, and the photometric activity proxies, the FliPer $F_{0.7}$, and the effective temperature become the most important predictors (fig. B.3, middle panel). Removing the *unknown* variables belonging to the Wav group (DS6) does not affect the performance of the model nor has impact on the importance of the predictors (fig. B.3, bottom panel).

If only predictors belonging to the *time series* group are used for the regression (DS7), the error in the photometric activity proxy obtained with the 20-day filter becomes the most important explanatory variable, followed by $S_{\rm ph}$, the corresponding error for the 55-day filter, and $S_{\rm ph}^{20}$. The $G_{\rm ACF}^{20,55}$ and $H_{\rm ACF}^{20,55}$ variables also appear in the top-10 list, being the former typically more important than the latter (fig. B.4, top panel).

When we use only astrophysical quantities (DS8), $F_{0.7}$ and F_7 become the most important predictors, followed by $T_{\rm eff}$ and the mass of the star (fig. B.4, bottom panel).

In the next section, we describe the procedure we followed to train XGBoost models from similar data sets to the ones we used for RFs, and present the results obtained.

* For regression RFs, the **ranger** R package computes the importance of the variables using the variance of the responses (Wright, Wager and Probst, 2019). The increase in a node purity is calculated from the reduction in the sum of the squared errors, whenever a variable is chosen for the split.

4.2 Results Obtained With Extreme Gradient Boosting

A popular technique in ML, *gradient boosting* (GB) is an ensemble approach able to convert weak learners into a strong joint model, by improving on top of the residual errors produced by the weak learners during the previous iterations. Given the popularity of GB—especially its optimised XGBoost algorithm—in the academic and business realms, we decided to train GB models using the XGBoost framework on top of the data sets presented in section 3.1, similarly to as we did for RFs in the previous section.

We started by preparing the data sets for XGBoost. Since our predictors and response are numeric, we did not need to encode categorical variables. In addition, even though XGBoost has been designed to work with sparse and missing data, we opted for using the major data set to which we previously applied data imputation, so that we could compare the performance of the RF and XGBoost models as fairly as possible.

Being an ensemble algorithm comprised of tree based learners, XGBoost is invariant to the scaling of the inputs and, hence, it does not require normalisation of its features (T. Chen and Guestrin, 2016). However, in order to avoid large scale differences between variables and optimise numerical computations, we opted to standardise the predictors, so to have all of them on the same scale.*

While ML algorithms are generally tolerant towards irrelevant or noise variables, too many irrelevant variables may originate models liable to overfit (Zumel and Mount, 2019). Removing predictors prior to modelling has the potential advantages of (*a*) decreasing the computational time and complexity of the resulting model, (*b*) removing some correlated features and, consequently, lead to more parsimonious and interpretable models, and (*c*) increasing the model performance by eliminating variables with degenerate distributions (Kuhn, Johnson et al., 2013). Given the large number of predictors of DS0 to DS4 and the results obtained in section 4.1, we dropped all predictors of the Wav group, and we cross-validated the features to the response, in an attempt to identify the most statistically significant variables and perform feature pruning prior to model training.†

To achieve this goal, we used the **vtreat** R package‡ (Zumel and Mount, 2016), in particular its `get_score_frame` function. It automates the estimation of the cross-validated significance of the predictors in single variable linear models with the response, and computes the R^2 of those models. Based on that information, it then recommends the set of predictors most suitable for model training, that is, it advises about which variables appear to be most likely to be useful to

* That decision also revealed helpful during the data set engineering, because it allowed us to detect particular cases of outliers with extreme and infinite values in some predictors (recall section 3.1). Infinite values became `NaN`s after scaling, which were easily detected after a summary of the statistics on the transformed variables.

† This pre-processing step might reduce collinearity, although it will not remove it completely because even if it eliminates some pairwise correlated variables, several predictors may be functions of two or more of the other variables.

‡ **vtreat** is an R package designed to prepare real-world data for supervised learning tasks (Zumel and Mount, 2019). Among others, **vtreat** automates tasks related to missing values in numeric variables, extreme or out-of-range values in numeric variables, missing values in categorical variables, and overfitting due to "nested model bias" and due to a large number of features. We were particulary interested in using **vtreat**'s ability to select the most suitable predictors for modelling out of a set of initial variables.

include in the model. We opted for **vtreat**'s conservative approach when recommending numerical variables, to reduce the possibility of mistakenly eliminate predictors that could be useful for the model. The approach consists in adopting a value of $1/n$ to the significance level of the linear models, where n is the number of predictors in the data set.

In the extreme case of having uniquely noise features, that are completely unrelated to the target variable, the significance levels of the one-variable models will be uniformly distributed in the $[0, 1]$ interval. The weakest possible significance threshold, allowing to eliminate as many noise predictors as possible, is $1/n$. This threshold assures that the expected value of the number of variables being recommend for modelling is 1 (Mount and Zumel, 2020), while relevant variables go through.

In spite of being conservative, this method does not prevent the elimination of variables that have a real but non-linear relationship with the response. We performed nonetheless several tests on the DS2, DS6, DS7, and DS8 data sets using XGBoost, and the models trained with the shorter recommended versions of the data sets always achieved at least the same (if not better) predictive results than the ones trained on top of their full counterparts.

The predictors that were removed in each data set typically correspond to the features in figs. 3.5 to 3.7 which were not correlated with the response variable. The total number of recommended predictors for each of the DS0 to DS8 data sets is indicated in table 4.3. For DS0, DS2, DS3, and DS4, the reduction of the number of predictors was approximately between 25 % and 30 %; for DS6 and DS7, that reduction was of 15 %; and for DS8, it was about 7 %.

Table 4.3: Number of predictors for the main data sets used during XGBoost training "before" and "after" the recommendation analysis.

Variables	DS0	DS2	DS3	DS4	DS6	DS7	DS8
Before	174	165	102	108	33	19	14
After	130	121	76	76	28	16	13

The **xgboost** package is able to deal with several types of input data, such as dense and sparse matrices. However, the authors recommend to use **xgboost**'s own class (`xgb.DMatrix`) for efficiency and speed (T. Chen and Guestrin, 2016), and so we converted all training and testing sets to *XGB dense matrices*.

Next, we performed a grid search to find the best set of values for the hyperparameters, as described in section 3.3.1. The results of the parameter optimisation are indicated in table 4.4. By comparing the results of table 4.4 with the original grid of section 3.3.1, we realise that:

- `colsample_bytree` oscillated within the three given values, so there is no tendency for the best value for this parameter, and the original possibilities or similar ones were kept in future grid searches;
- `eta` remained below 0.1, and so smaller values were prioritised; this comes with the cost of longer convergence times and danger of overfitting, hence we considered the `gamma` hyperparameter in future grid searches for a more conservative approach;

- `max_depth` was always equal to 5; given that the grid offered only two possibilities (3 and 5), cases with larger values (6 and 7) were looked into in subsequent models; and
- `subsample` remained above 0.60, so we investigated initial values between 0.60 and 1 in future grid searches.

Table 4.4: Best set of values for the XGBoost hyperparameters after performing grid search. The `nrounds` parameter indicates the number of iterations to be carried out by the XGBoost algorithm, in order to achieve the best model when using the optimal values of the hyperparameters for the corresponding data set.

Hyperparameter	DS0	DS2	DS3	DS4	DS6	DS7	DS8
`colsample_bytree`	1/3	1/3	1/2	1/3	2/3	1/2	2/3
`eta`	0.08	0.03	0.05	0.04	0.07	0.05	0.03
`max_depth`	5	5	5	5	5	5	5
`subsample`	0.85	0.60	0.80	0.85	0.80	0.85	0.65
`nrounds`	92	417	336	487	1000	712	98

The models were built using the optimal sets of the hyperparameters found by the grid search, and the algorithm was stopped after `nrounds` iterations, as indicated in the last row of the table. We had set the maximum number of possible iterations to 1000 during the learning phase, but that never corresponded to the optimal value of `nrounds`, except for DS6. All other data sets required less iterations to build the best model according to the specific set of optimal hyperparameters and, apart from DS7, they required less than 500 iterations.

The performance of the models obtained by applying XGBoost to the data sets of tables 4.3 and 4.4 to train models to predict stellar rotation periods is compiled in table 4.5. Similarly to table 4.2, the number of predictors used to train the models is indicated below each data set, on the header of the table. The quantities used to assess the quality of the models are the same as in the aforementioned table.

Overall, the XGBoost models were wrong, on average, approximately between 12.3 to 38.4 % of the time, as indicated by μ_{err}. Notably, the models trained with DS2 and DS3 exhibit the same mean relative absolute value of the residuals, indicating that, for the XGBoost models, and contrarily to the RF ones, the quality is not affected when the CS are suppressed from the predictors. The quality of the model is not substantially affected when both CS and GWPS variables are removed, even when only predictors of the TS type are present in the data set. The turning point is when all variables except the astrophysical ones are removed. In the presence of uniquely classical astrophysical predictors, the model performs poorly, being wrong, on average, 38.4 % of the times, making it impractical for predictive tasks.

The accuracies also decrease as the number of predictors is reduced, but the largest reduction—approximately 33 points for acc_{10}—is seen when both CS and GWPS variables are removed, *i.e.*, when going from DS2 to DS6. The accuracies decrease about 5 points when only TS variables are present, and approximately 22 points—to 12 % on acc_{10}—when only astrophysical predictors are

used to train the model. The difference in the acc_{10} accuracy in the first four data sets (containing the CS and GWPS predictors) is of about 7 points, and 27 points in the three last data sets (without those predictors). However, unless we are willing to accept errors up to 20 %, the interval accuracies remain below 90 % for all models.

Table 4.5: Quality assessment of the XGBoost models according to the number of predictors. All quality measures are presented with three significant figures, except the adjusted-R^2 (with four). The top row indicates the number of predictors in each data set. The first row shows the mean absolute value of the relative residuals; the three following rows correspond to the interval-based accuracies defined in section 3.3.3, with widths of 20, 10 and 5 % of the reference values; the last three rows present the root-mean-squared error and the mean-absolute error, both computed on the training and testing sets, and the adjusted coefficient of determination (calculated on the testing set). The best results obtained with the metrics (except for RMSE and MAE) are marked in bold.

Assessment	DS0 130	DS2 121	DS3 76	DS4 76	DS6 28	DS7 16	DS8 13
μ_{err}	**0.123**	0.145	0.145	0.157	0.153	0.170	0.384
$\textbf{acc}_{\textbf{20}}$ (%)	**89.4**	87.7	86.8	86.2	66.4	59.4	25.1
$\textbf{acc}_{\textbf{10}}$ (%)	**79.5**	77.4	76.1	73.0	39.6	34.1	12.2
$\textbf{acc}_{\textbf{5}}$ (%)	**67.5**	62.1	58.8	64.1	21.3	18.7	6.29
RMSE (train/test)	1.79 2.82	1.61 2.79	1.37 2.91	1.32 3.08	1.05 3.63	2.07 3.96	7.08 7.39
MAE (train/test)	0.807 1.06	0.795 1.14	0.737 1.18	0.758 1.33	0.784 2.24	1.54 2.57	5.27 5.41
$\textbf{R}^2_{\textbf{adj}}$ **(test)**	0.9473	**0.9482**	0.9429	0.9367	0.9076	0.8907	0.6335

The RMSE on the testing set decreased as the number of predictors increased, except when going from DS2 to DS0; in turn, the RMSE on the training set initially decreased when going from DS8 to DS6, but it then increased until DS0. The difference between the training and testing RMSE varied within the $]\,1, 2\,[$ interval for the DS0 to DS7 data sets, indicating some level of overfitting in those models, but that difference was the smallest (0.31) in the model built with DS8. Therefore, although it presented the poorest prediction performance, the DS8 model was the one exhibiting the smallest level of overfitting.

The model hielding the best goodness of fit, as measured by R^2_{adj} on the testing set, was DS2—we note that DS2 is a data set deprived of Prot variables—closely followed by DS0, DS3, and DS4, all with values approximately between 93 % and 95 %. The adjusted R-squared for DS6 and DS7 was close to 90 %, and for DS8 it was of about 63 %. We compared these R^2_{adj} values with the corresponding unadjusted coefficients of determination and the ones of the models trained in section 4.1, and they are all of the same magnitude within each model, with differences typically on the third decimal place (when written as real numbers). Therefore, it is safe to interpret R^2_{adj} as the fraction of the response's variation that is explained by the predictors.

The scatter plots of the real rotation periods *vs*. the predicted ones are displayed in fig. 4.2 and, with more detail, in figs. B.11 and B.12, in the appendix. Similarly to fig. 4.1, the blue solid lines indicate the identity function, and red dashed lines represent the linear models between the predicted and the real values. All graphs evince a good correlation between the predicted and the real values of the stellar rotation periods. Apart from DS8, the dispersion is not considerably large.

Some outliers can be identified, both in the regions of small and large rotation periods, but they are not in great number. Although the majority of the stars in the data sets have rotation periods below 50 d, similarly to the RFs cases, the XGBoost models seem to struggle to accurately predict stellar rotation periods beyond the interval $[5, 50]$ days, approximately. Together with the results obtained for the RFs models, this confirms our suspicions that stars with rotation periods below 7 d and above 45 d are not reliable examples to train a model, and together they validate the approach described in section 3.3.2 for building the DS9 and DS10 data sets. Moreover, the outliers do not seem to be predicted by the linear models, as is evident from the dashed red lines. Therefore, we can conclude that the adjusted R-squared coefficients of table 4.5 are not inflated.

The marginal experimental distributions for the predicted and the real rotation periods, as highlighted by the histograms and density plots of figs. B.11 and B.12, are visually very similar to each other on every model, in terms of centrality, dispersion, and kurtosis. The coefficients of determination between the predicted and the reference values reported in the figures do not differ significantly from R^2_{adj} indicated in table 4.5, and the F-tests and related p-values indicate that the sets of explanatory variables are statistically significant, as a whole.

The most important variables for the models are highlighted in the bar plots of figs. B.5 to B.7. Apart from the Prot features, which naturally dominate in terms of importance when they are present in the data set, CS, GWPS, and TS (mostly ACF parameters) are the most important variables for the prediction of rotation periods.

In the model created from DS0, the first four most important variables are rotation periods obtained from the three aforementioned methods (GWPS, ACF, and CS) for the 55- and 20-day filters. The central periods of the Gaussians fit to both GWPS and CS are also important, as well the photometric activity proxy and the effective temperature.

For DS2, the central periods and standard deviations of the Gaussians fit to GWPS and CS computed for the 20- and 55-day filters are the most important features. The only variable in the top-10 not belonging to this category is the photometric activity proxy. The variables were clustered in terms of importance, so that predictors with the same bar colour have importances of similar magnitude. The variable that stands out in the model created with DS2 is `gwps_gauss_2_-1_55`. Although the algorithm groups the remaining eight variables in the same group, we would put the following five predictors in one group, and the last four in another cluster.

In DS3, the central period computed on the Gaussian fit to the GWPS for the 55-day filter is the most important variable, followed by the central periods computed on the 80- and 20-day filters (both were clustered in the same group), and then the remaining predictors, all in another cluster (mainly photometric proxies, effective temperature, and logarithm of the surface gravity).

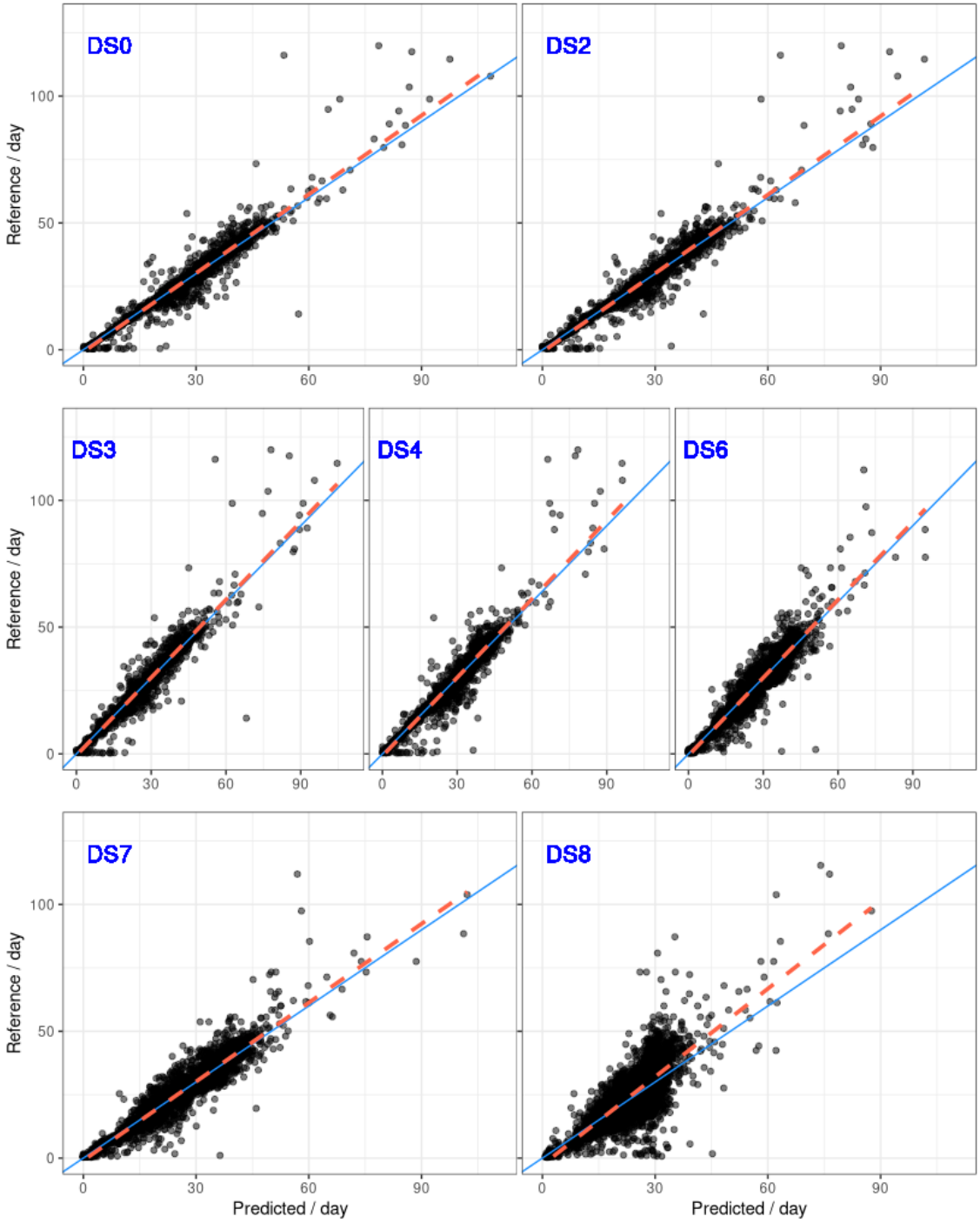

Figure 4.2: Scatter plots of the reference rotation period as a function of the predicted values for XGBoost models built with the data sets DS0 to DS8. The blue solid lines indicate the identity function, and the red dashed lines represent the linear model between the reference and the predicted values.

For the model learnt from DS4, the central periods and standard deviations of the Gaussians fit to the CS for the 55- and 20-day filters were the most important variables, followed by the FliPer value for $\nu_C = 0.7$, central period for the 80-day filter, the photometric activity proxy, the effective temperature, and the amplitude of the period of the Gaussian with the highest amplitude fitted to the CS for the 20-day filter. Two clusters can be identified in terms of importance, the border of which lying between the aforementioned central periods and $F_{0.7}$.

After removing the CS and GWPS variables, seven out of the ten most important variables are of the TS type, and only three are astrophysical in nature. Three importance clusters can be identified: (*i*) one containing the error of the photometric activity proxy computed on the ACF for the 20-day filter, and the 0.7-FliPer value; (*ii*) another composed uniquely of the photometric activity proxy; and (*iii*) a third one with the effective temperature, the ACF photometric activity proxy for the 20- and 55-day filters, the height of the period of the highest peak in the ACF ($P_{\rm ACF}$) at a lag greater than zero for the 20-day filter, the error of the photometric activity proxy, the mass, and the mean difference between the height of $P_{\rm ACF}$ and the two local minima on both its sides for the 20-day filter.

Considering only the TS variables (DS7), the photometric activity proxies and their errors occupy a prominent position in the importance list. Two to three clusters are identified, being $S_{\rm ph}^{\rm ACF,\,err}$ for the 20-day filter the most important variable. We can state that a first cluster is composed of this variable; a second contais $S_{\rm ph}^{\rm ACF,\,err}$ for the 55-day filter, $S_{\rm ph}$, and $S_{\rm ph}^{\rm ACF}$ for the 20-day filter; and a third one is composed of $G_{\rm ACF}$ for the 20-day filter, $S_{\rm ph}$, $S_{\rm ph}^{\rm ACF}$ for the 55-day filter, $H_{\rm ACF}$ and $G_{\rm ACF}$ for the 55-day filter, and $H_{\rm ACF}$ for the 20-day filter.

As for the astrophysical predictors alone, $F_{0.7}$ is clearly the most prominent predictor in terms of importance. It is followed by the effective temperature and F_7 in a second cluster, and a third cluster containing the mass, F_{50}, F_{20}, the magnitude from the *Kepler* catalogue, $\log g$, and the lower and upper errors for the effective temperature. When it comes to classical astrophysical variables, the Flicker in Power values and the effective temperature are the features that contribute the most to the prediction of rotation periods.

In view of the results obtained with the batch of models trained with RFs and XGBoost, we decided to create a new data set, DS9, containing only the most relevant variables—as identified while training the aforementioned models—and only stars with rotation periods in the range 7 d to 45 d, as explained in section 3.3.2. DS9 was further refined in order to include only the top-10 most important variables for the models learnt from it, so we could examine how much predictions of stellar rotation periods are affected by removing less important variables, and how much of the previous results would hold with such a stripped down data set. The results are going to be presented in the following section.

4.3 Minimising the Size of the Data Sets

In section 3.3.2, we described how we built two new data sets of reduced size, DS9 and DS10, from the master data set stripped out of Prot variables (DS2). Using those data sets, we trained RF

and XGBoost models, in a similar way to what we have done before with DS0, DS1, ..., DS8. In what follows, we will describe the procedure we followed and the results we obtained.

4.3.1 Results From Models Trained With DS9

DS9 resulted from selecting the most important predictors for all the models trained so far with both RFs and XGBoost, and by filtering out stars with rotation periods outside the interval $]\,7, 45\,[$ days. We were left with a data set comprised of 36 predictors and 13 627 instances, which was split into training and testing sets in a 80-to-20 ratio.*

We started by training two RFs models, one on top of the full DS9 data set, and another after performing feature pre-pruning, by selecting the most statistically significant predictors, as we previously did for the XGBoost models. We wanted to check out if the predictive performance would be affected by applying this procedure in a data set of reduced size, such as DS9. The pruned version of DS9 contained 33 predictors (three less than the full set). In both cases, we searched for the optimal values of the `mtry` and `min.node.size` hyperparameters using the same grid as before, but we fixed `splitrule` to "variance", since this was always found as the best value in the previous trainings, which allowed us to save extra learning time. The optimal number of randomly sampled predictors to possibly split at in each node, `mtry`, was 15 in the full version of DS9, and 13 in the set with recommended predictors, while the optimal minimal node size was 2 for both of them. The training time, given that we were always using a shared machine with heavy traffic, was similar to the training time obtained before with DS6, which has approximately the same number of features as DS9.

The quality assessment of the RFs models trained with DS9 are presented in the first two rows of table 4.6. Overall, the performance of the model built with the recommended variables is at the same level as the performance of the model trained with all predictors available in DS9. On average, the models were wrong about 2.3 % of the time. The 10 % interval-based accuracy was above 90 %. The RMSE measured on the testing set was about 2.7 times larger than its training counterpart, and the testing MAE was approximately 2.8 times larger than the traning one. These differences, alghouth not significantly large, denote some degree of overfitting by the models.

The goodness of fit, as measured by R^2_{adj}, was circa 96 %, denoting high quality models, in the sense that, on the one hand, they surpass the performance of all the models previously built with RFs and XGBoost and, on the other hand, they point to a large fraction of the variability of the response explained by the predictors.

The scatter plots of the reference stellar rotation periods *vs*. the predicted values for the model built with the recommended variables are illustrated in fig. 4.3.† The corresponding residuals and 10 %-error metric plots are illustrated in fig. 4.4.‡ Overall, these plots indicate that there is a general good agreement between the predicted and the reference response values. This is reinforced by the marginal histograms and density plots for both the predicted and reference values on the right

* Given the reduction in the number of cases when filtering out stars, we decided to adjust the relative sizes of the training and testing sets.

† Equivalent plots for the model built with all the features available in DS9 can be found in the appendix, in fig. B.10.

‡ The equivalent plots for the model built with all variables available in DS9 are illustrated in fig. B.13.

panel of fig. 4.3, which are similar between them. Most of the points fall on the $y = x$ line, but some outliers can be seen, both representing under and over predicted cases. The largest errors can go up to approximately 20 d in magnitude. The relative errors oscillate between nearly -1 and $+0.5$, corresponding to overpredictions of approximately twice and underpredictions of about half the real values, respectively. Except for two cases, the underpredictions are always larger than 50 % of the corresponding real values. The overpredictions can go up to 200 % of the true values, the largest of which typically lie in the 20 d to 40 d range. The situation is similar for the model built with all available variables of DS9.

Table 4.6: Quality assessment of the two RF (first two rows) and the XGBoost (last row) models learnt with DS9. The indices "rec" and "all" indicate the subset of recommended variables and the full version of DS9, respectively. The XGBoost model was built upon the recommended set of variables. The considerations applied to the quality metrics in table 4.2 hold here.

DS9	$\mu_{\mathbf{err}}$	$\mathbf{acc}_{20}$	$\mathbf{acc}_{10}$	$\mathbf{acc}_{5}$	**RMSE** **train** **test**	**MAE** **train** **test**	$R^2_{\mathbf{adj}}$
$\mathbf{RF}_{\mathbf{rec}}$	0.0231	0.945	0.902	0.837	0.684 1.84	0.212 0.597	0.9601
$\mathbf{RF}_{\mathbf{all}}$	0.0229	0.946	0.901	**0.839**	0.681 1.84	0.210 0.592	0.9597
XGB	**0.0226**	**0.958**	**0.906**	0.810	0.921 1.45	0.389 0.564	**0.9758**

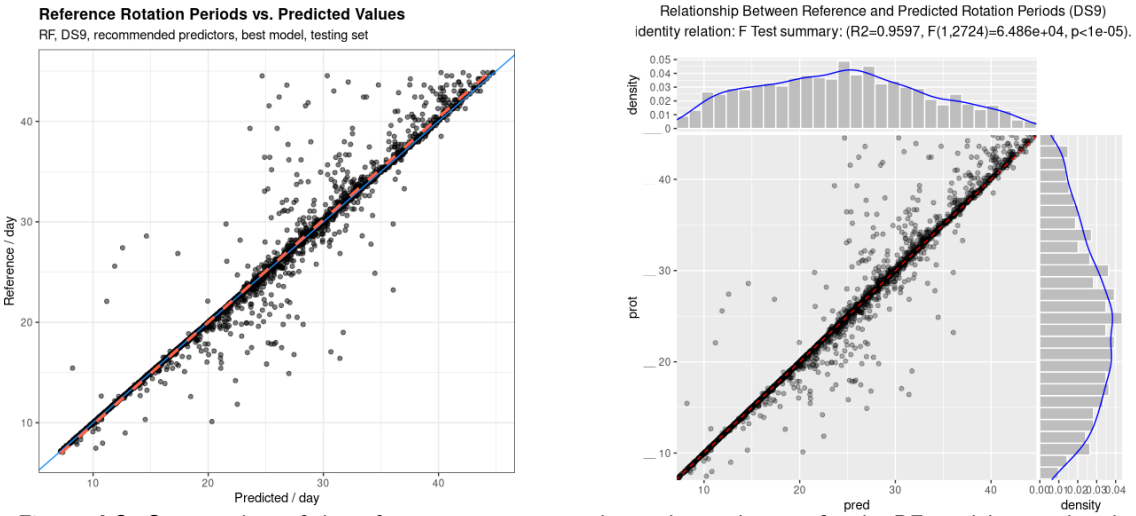

Figure 4.3: Scatter plots of the reference rotation periods *vs.* the predictions for the RF model trained with the DS9 data set using recommended variables. On the left panel, the blue solid line represents the identity function, and the red dashed line the linear model between the predicted and the true values. On the right panel, the red dashed line refers to the identity function; the margins contain the histograms and density plots of the sample of predicted and reference rotation periods.

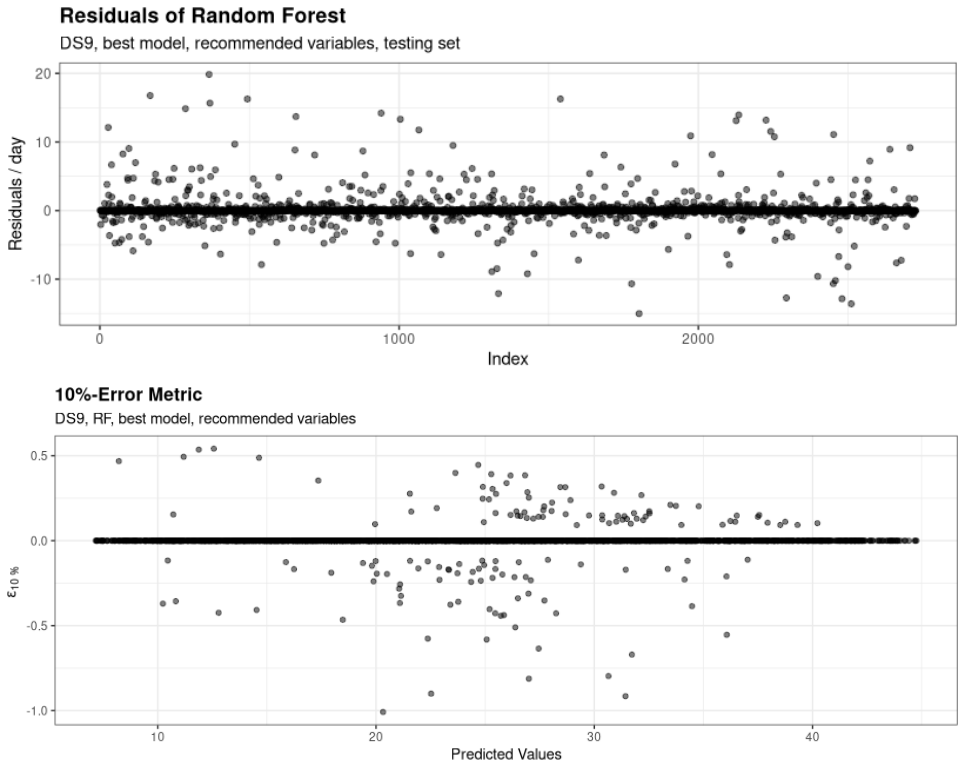

Figure 4.4: Residuals (top panel) and 10 %-error metric (bottom panel) for the RF model trained with the recommended variables in DS9.

Figure 4.5 illustrates all predictors sorted by the degree of importance as they were used to create the RF models. The panel on the top refers to the model learnt from all the variables available in DS9, while the one on the bottom is related to the model trained with the recommended predictors. We can identify five to six clusters in terms of importance, with the two most important variables being `cs_gauss_2_1_55` and `gwps_gauss_2_1_55`. The set of the top-10 most important variables is the same in both data sets.

The XGBoost model was trained using uniquely the recommended variables of DS9. Taking advantage of the reduced size of the data set, and in face of the set of optimal values for the hyperparameters of table 4.4, we decided to increase the number of values available for the grid search as follows:

- `colsample_bytree`: $\{1/2, 2/3, 3/4\}$;
- `eta`: $\{0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1\}$;
- `max_depth`: $\{5\text{L}, 6\text{L}, 7\text{L}\}$;
- `subsample`: $\{0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 1.0\}$;
- `gamma`: $\{1, 10, 30, 100\}$;
- `min_child_weight`: $\{1\text{L}, 10\text{L}, 50\text{L}\}$.

This gave rise to 7776 models without counting with the 10CV with five repetitions. However, since the learning was interrupted for any model whenever the performance was not improving for

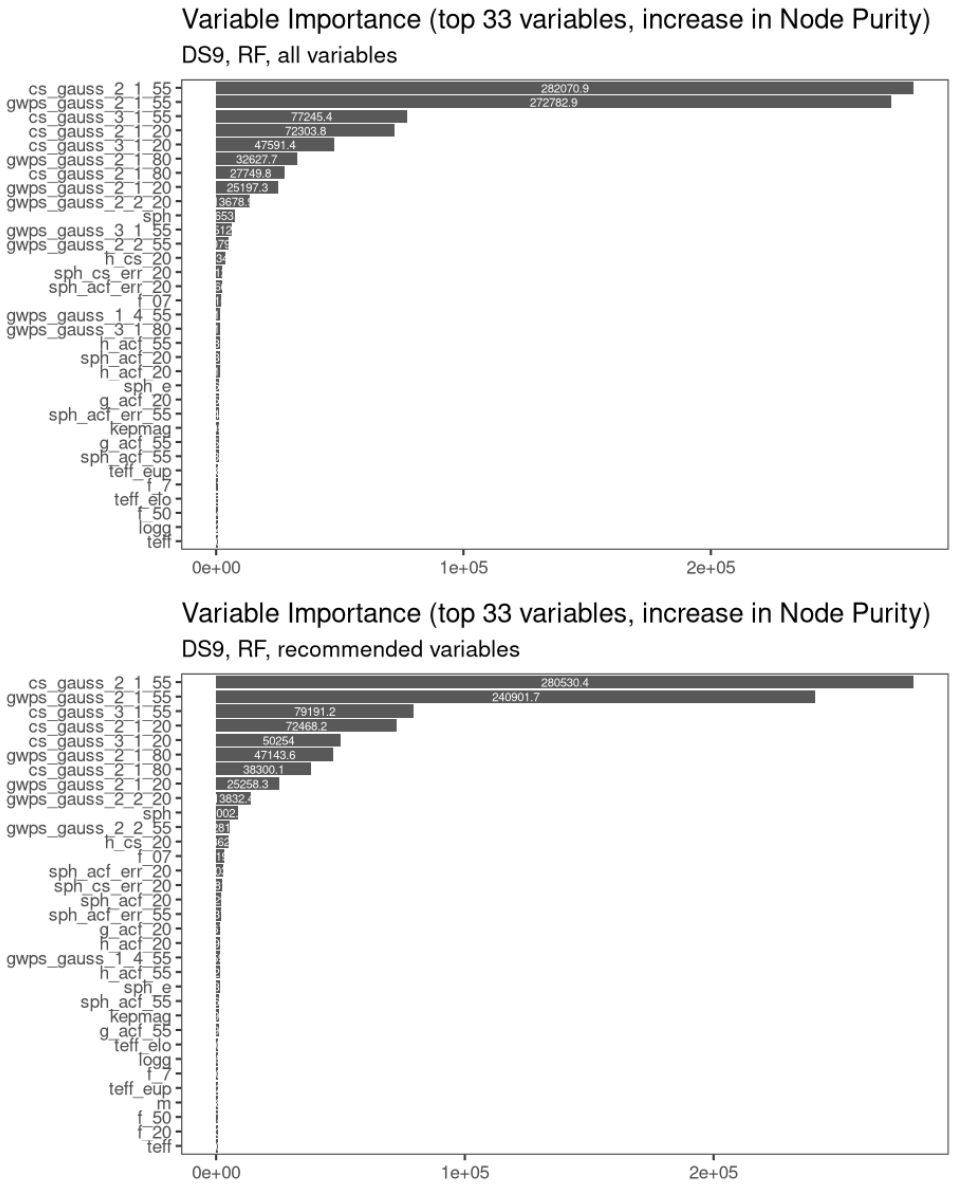

Figure 4.5: Variable importance in the RF models created with DS9 using all available variables (top panel) and the set of recommended variables (bottom panel).

five rounds, the grid search took approximately six hours to finish (about three times more than the previous grid search on the XGBoost model built on top of DS6), making the model learning still feasible in a timely manner in light of current common computing resources.* The best set of values for the hyperparameters was the following:

- `colsample_bytree`: 1/2
- `eta`: 0.04
- `gamma`: 1
- `max_depth`: 7
- `min_child_weight`: 10
- `subsample`: 1

* We used a shared machine with 40 Intel Xeon Silver 4214 CPUs running at a clock frequency of 2.20 GHz. We split the search into four grids, each with 1944 points, and executed them in parallel using at most 10 CPUs each.

The optimal model was achieved for 153 iterations of the XGBoost algorithm. We note that (*a*) the optimal `max_depth` was 7, the maximum corresponding value in the grid, similarly to what happened before, which suggests that the algorithm tends to choose the largest value of this parameter, *i.e.*, deeper trees; and (*b*) this time we allowed for the possibility of regular boosting, and the best results were obtained with `subsample` equal to 1. While deeper trees and regular boosting might increase the chances of overfit, the `gamma` hyperparameter makes the model more conservative and introduces an extra layer of regularisation, which can compensate for that.

The quality assessment of the results obtained with the XGBoost model is compiled in the last row of table 4.6. As in the RF case, the model was wrong, on average, approximately 2.3 % of the time. The 10 %-interval accuracy was above the 90 % mark, and approximately 0.5 points better than the RF counterparts. The differences between the training and testing RMSE and MAE were less than the half of those in the RF cases, indicating less overfitting of the XGBoost model than its RF counterparts. The goodness of fit, as measured by the adjusted coefficient of determination, was better than in the RF models, given that circa 97.6 % of the variability of the response is explained by the predictors. The MSE for this model is dominated by the variance. These assessment metrics indicate a solid yet computationally cheap model, with great predictive power.

Figure 4.6 presents the scatter plots of the real stellar rotation periods *vs*. the predicted ones for the XGBoost model built with DS9 using the recommended variables, and fig. 4.7 illustrates the corresponding residuals and 10 %-error metric plots.

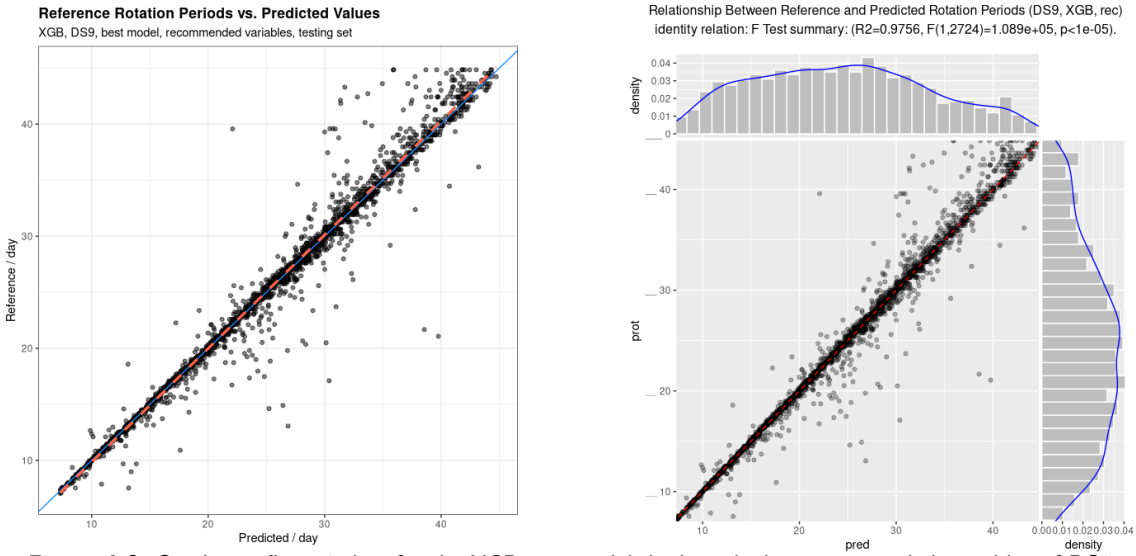

Figure 4.6: Similar to fig. 4.3, but for the XGBoost model, built with the recommended variables of DS9.

Similarly to the RF models trained with DS9, the scatter plots, and the marginal histograms and density curves indicate an overall good agreement between the predicted and the real values of the stellar rotation periods. The F-test and the corresponding p-value of the right hand panel of fig. 4.6 indicate that the predictors are jointly significant. Most of the values occupy a position close to the $y = x$ line, with few outliers. Nevertheless, some of the outliers have magnitudes differing approximately 20 d from the real values, indicating that the model, although rarely, can go way off the true values. Similarly to the RF sibling models, the relative error oscillates approximately

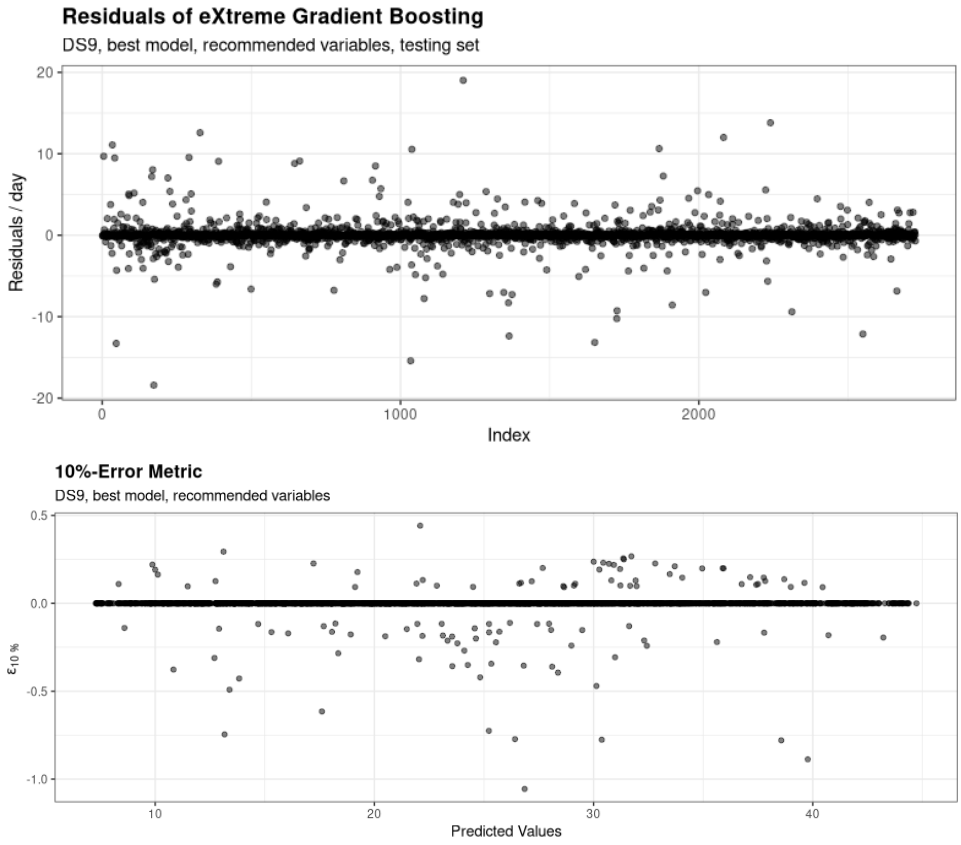

Figure 4.7: Similar to fig. 4.4, but for the XGBoost model.

between -1 and $+0.5$, indicating that the model can overpredict periods as large as twice the real values, and underpredict rotation periods as low as half of the reference values. There are no underpredictions less than 50 % of the true values. Six of the overpredictions are larger than 175 % of the reference rotation periods.

The predictors of the model are sorted out by degree of importance in fig. 4.8. The algorithm in the **xgboost** package identified five clusters. The first of them is the most prominent, composed of the variables `cs_gauss_2_1_55` and `gwps_gauss_2_1_55`, similarly to what happened with the RFs models trained with DS9. The top-10 variables are the same as for the RF counterparts, although their sorting order vary slightly.

Given the relevance of the top-10 most important variables in the three aforementioned models built with DS9, we explored further the possibility to reduce the number of predictors without loosing too much predictive power. We created one last data set, DS10, composed of the same cases of DS9, but with only the ten most important predictors.

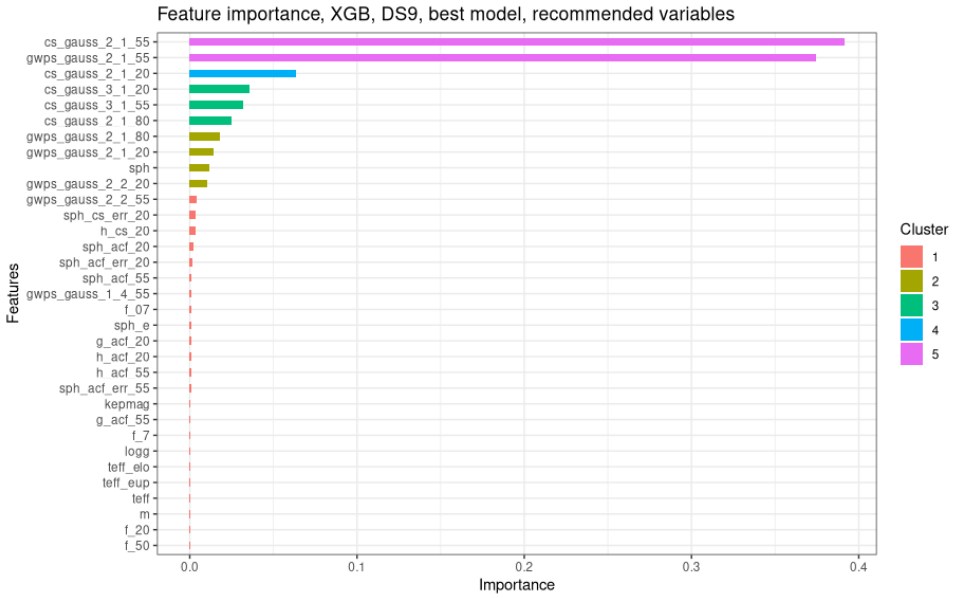

Figure 4.8: Variable importance in the XGBoost model created with DS9 using the set of recommended variables.

4.3.2 Results From Models Trained With DS10

We trained a RF and a XGBoost model, performing a grid search for the optimal values of the hyperparameters. In the case of the latter, we took advantage of the small size of the data set and refined the grid, as we will describe below.

For the RF model, we kept the same grid as for DS9, that is, with `splitrule` equal to "variance", and `mtry` and `min.node.size` varying within the set of values indicated in section 3.3.2. The training took approximately 20 min to finish on the same machine used for learning the previous models. The quality assessment of the RF model trained with DS10 is encompassed in the first row of table 4.7. The results for the RF model are similar to the ones obtained for its cousins trained with DS9. The most notorious difference is that the DS10 version was able to explain 1 % less of the variability of the response than their DS9 RF counterparts.

The scatter plots of the real rotation periods *vs*. the predictions for the RF model are depicted in fig. 4.9, and the corresponding residuals and 10 %-error metric are illustrated in fig. 4.10. The scatter plots, the marginal histograms, and the marginal density plots evince a strong correlation between the predicted and the real values of the rotation periods. Most of the points lay on the line representing the identity function (solid or dashed blue lines), or close to it. The largest errors can go up to about 20 d in magnitude and, similarly to what happened for the DS9 models, the relative errors oscillate between approximately -1 and $+0.5$, indicating nearly double overpredictions and half underpredictions, respectively. While the overpredictions occur through the whole range of predicted values, underpredictions below 50 % of the real value happen on the range 20 d to 40 d. Notably, the model is depleted of overpredictions above 200 %, approximately, meaning that, at most, it predicts the double of the real value. This requires further investigation, since this upper

limit may be an indication that, in the worst-case scenario, the model captures the first harmonic of the rotation period.

Table 4.7: Quality assessment of the RF model (first row) and the XGBoost models (second and third rows) trained with DS10. The three models were learnt from the ten most important variables of the models trained with DS9. The considerations applied to the quality metrics in tables 4.2 and 4.6 hold here.

DS10	$\mu_{\mathbf{err}}$	$\mathbf{acc}_{20}$	$\mathbf{acc}_{10}$	$\mathbf{acc}_{5}$	**RMSE** **train** **test**	**MAE** **train** **test**	$R^2_{\mathbf{adj}}$
RF	**0.0222**	0.945	**0.913**	**0.860**	0.757 2.00	0.214 0.583	0.9525
$\mathbf{XGB_1}$	0.0314	0.951	0.901	0.744	1.45 1.79	0.707 0.797	**0.9652**
$\mathbf{XGB_2}$	0.0238	**0.955**	0.910	0.832	0.979 1.73	0.405 0.601	0.9651

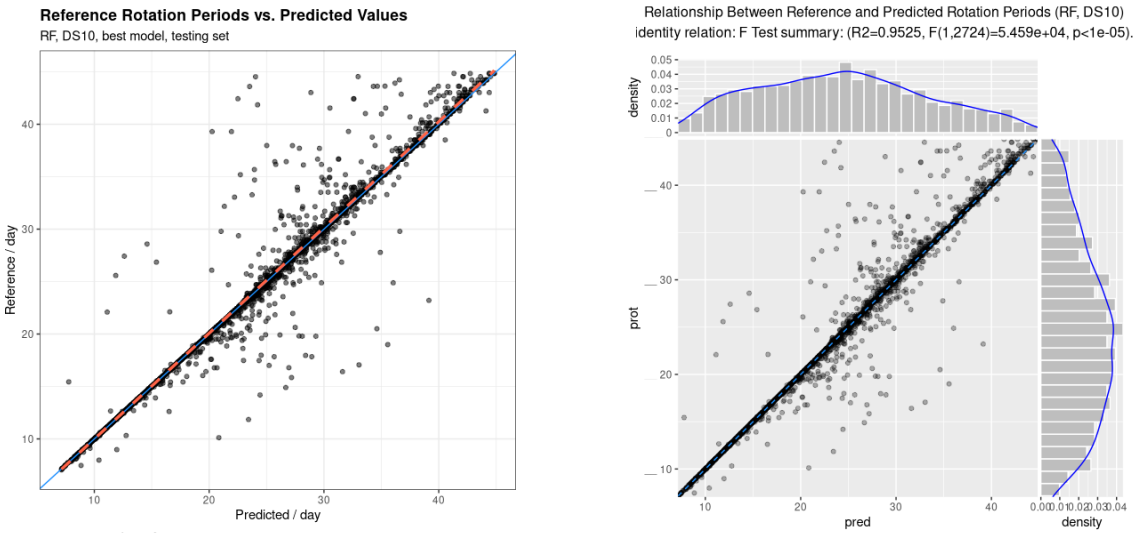

Figure 4.9: Scatter plots of the reference rotation periods *vs.* the predictions for the RF model trained with the DS10 data set. On the left panel, the blue solid line represents the identity function, and the red dashed line the linear model between the predicted and the true values. On the right panel, the blue dashed line refers to the identity function; the margins contain the histograms and density plots of the sample of predicted and reference rotation periods.

In the case of the XGBoost, we started to perform a search for the optimal values of the hyperparameters using the same grid as for DS9, with a 10CV with five repetitions. The model took about four hours to train. The best values found for the hyperparameters were the following:

- `colsample_bytree`: 2/3
- `eta`: 0.05
- `gamma`: 1
- `max_depth`: 7
- `min_child_weight`: 10
- `subsample`: 0.9

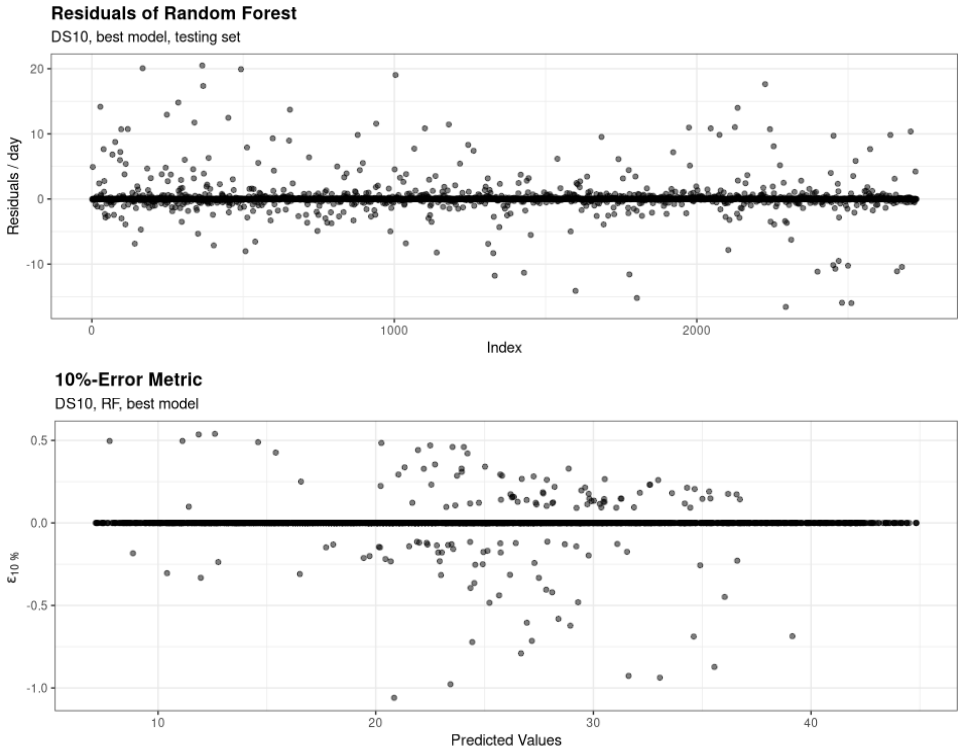

Figure 4.10: Residuals (top panel) and 10 %-error metric (bottom panel) for the RF model trained with the DS10 data set.

The best value for `max_depth` corresponded to the maximal one available in the grid. In the case of `subsample`, contrarily to the DS9 case, the model was optimised with a stochastic sampling of 90 % of the total cases, instead of a regular boosting. The optimal XGBoost model was built after 19 rounds of the algorithm. Its quality assessment can be inferred from the metrics' values of the second row of table 4.7. Overall, the quality metrics are in par with the ones from its RF sibling. However, we notice (*a*) a slight decrease in the 10 % accuracy, (*b*) a slight increase in the adjusted R-squared, and (*c*) less overfitting of the model (the differences between the training and testing values on both RMSE and MAE are smaller than in the RF case).

Finally, we went one step further, and we took advantage of the fact that the data set was stripped off of most predictors to refine the grid search. Hopping to find the best set of values for the XGBoost hyperparameters, the size of the grid was increased according to the following values:

- `colsample_bytree`: $\{1/3, 1/2, 2/3, 3/4\}$;
- `eta`: $\{0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1\}$;
- `gamma`: $\{1,\ 10,\ 30,\ 100\}$;
- `max_depth`: {3L, 4L, 5L, 6L, 7L};
- `min_child_weight`: $\{1,\ 10,\ 30,\ 50\}$;
- `subsample`: $\{0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90\}$.

This was the most ambitious grid search we carried out during this project, giving raise to 28 160 models, without taking into account the 10CV with five repetitions. On the same machine used for DS9, the training process took approximately 75 h to complete, given that the CPUs were shared between several users and this time we did not parallelise the learning by subdividing the grid, allowing the boosting to be carried out by 30 CPUs simultaneously, instead of the previously 10. The set of best values for the parameters was the following:

- `colsample_bytree`: 3/4
- `eta`: 0.05
- `gamma`: 1
- `max_depth`: 7
- `min_child_weight`: 1
- `subsample`: 1

The optimal model was obtained after 110 rounds of the algorithm, when applying the values above to the hyperparameters. Its quality can be assessed using the values on the third row of table 4.7. On average, the DS10 XGBoost model was wrong 2.4 % of the time, as indicated by the relative mean of the absolute values of the residuals. The 10 % interval-accuracy is in par with the ones from its siblings (RF and XGBoost), equal to 91 %. The scatter plots of the real *vs*. predicted values are illustrated in fig. 4.11.* There is a good linear agreement between the predicted and

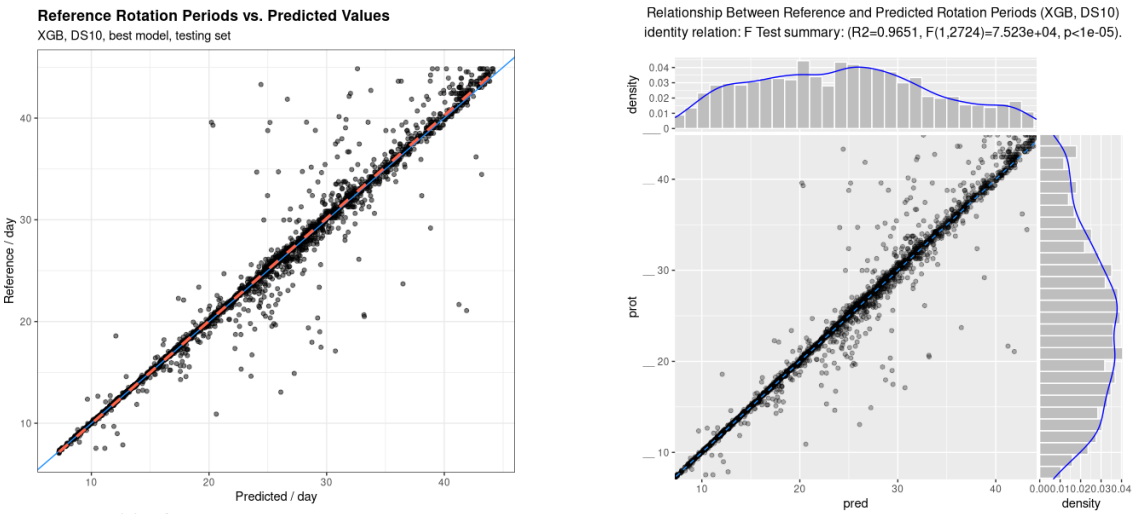

Figure 4.11: Scatter plots of the reference rotation periods *vs*. the predictions for the XGBoost model learnt from DS10. The blue solid and dashed lines, respectively on the left and right panels, represent the identity function, while the thicker red dashed line on the left plot represents the linear model between the predictions and the real values.

the real values: the linear model is not being affected by the outliers, and the marginal histograms and density plots of the right hand panel are very similar between the predictions and the real values. The residuals are mostly centred on the horizontal line $y = 0$ but, as in the previous models, they can attain magnitudes of about 20 d in absolute value. The bottom panel of fig. 4.12 shows that when we are willing to accept a 10 % error, most of the residuals are zero. Overpredictions never go beyond about the double of the corresponding real value. Notably, approximately the half

* There are only minute differences between these graphs and the scatter plots for the XGB_1 model, so the analysis is similar to the one we present here.

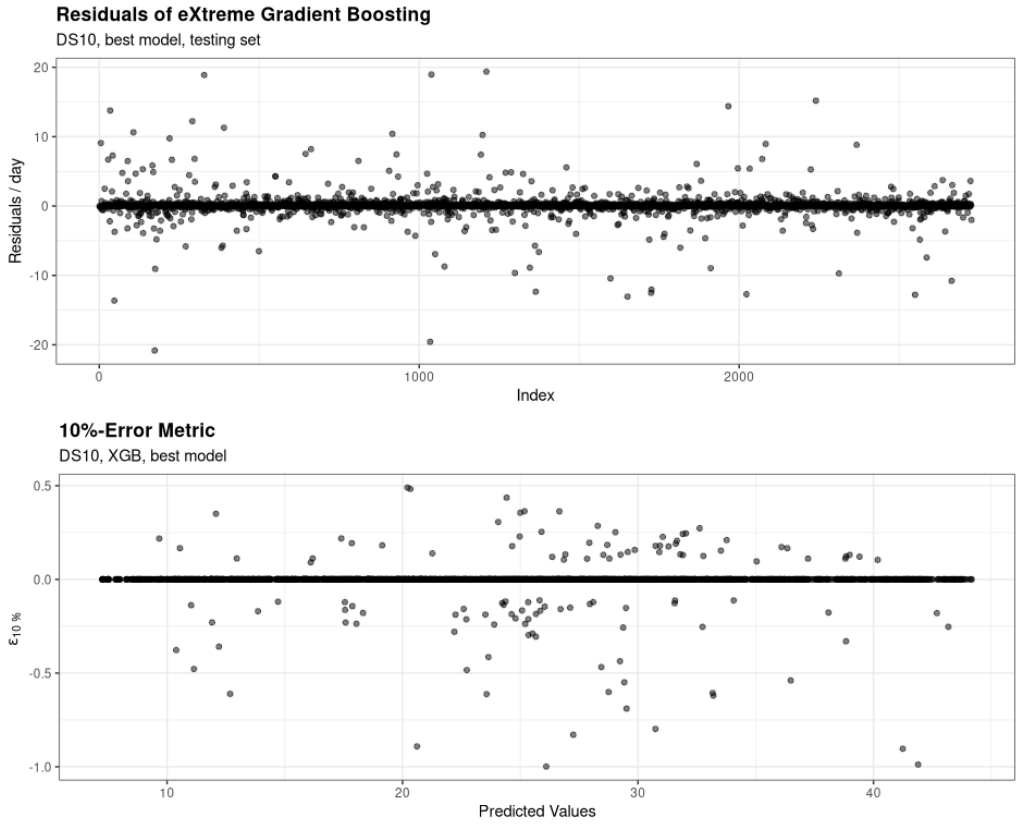

Figure 4.12: Residuals (top panel) and 10 %-error metric (bottom panel) for the XGBoost model trained with DS10. There are some outliers with magnitudes up to 20 d.

and the double of the real values are sometimes captured by the model. The largest overpredicted values occur for rotation periods above 20 d, approximately.

The variability of the response is mostly explained by the predictors, as can be attested by the plots and a goodness of fit of $R^2_{\text{adj}} = 0.9651$. This model overfits less than its RF sibling, since the differences between the training and testing RMSE and MAE are smaller in this case. The different levels of overfitting might explain the differences in interval-accuracies and adjusted R-squared between the models.

In the following section, we will discuss the results portrayed so far, and will present a broad view of the main ideas extracted from them.

4.4 Discussion of the Results

In sections 4.1 to 4.3, we trained several models using the data sets from DS0 to DS10, which in turn were built out of the S19 and B21 data sets. In doing so, we optimised the sets of hyper-parameters for each method, using a thorough grid search approach.

The results indicate that the two most important variables to predict rotation periods of K and M stars out of the *Kepler* catalogue are arguably `cs_gauss_2_1_55` and `gwps_gauss_2_1_55`.

After Prot features being removed from the data set, these variables always made up to the top-10 in terms of importance, provided they were available for training, and they stood out from the rest of the features. They correspond to the central period of the first Gaussian fitted to the CS and GWPS for the 55-day filter, respectively. The predictive performance of the RF and XGBoost models degrades significantly when these features are removed from the data set. Not only those two predictors, but most of the CS and GWPS features are relevant for the predictive performance of the models, as it is attested by the results from DS10. Therefore, CS and GWPS predictors based on the central period of the fit Gaussian should be available in the data set used for the learning process.

However, we believe that the level of importance attained by the CS and GWPS families of variables is related to the fact that most of the reference stellar rotation periods were estimated from them. Hence, there should be a strong correlation between those features and the response. A way to test this hypothesis would be to train new models from structured data obtained from synthetic light curves, created with reliable stellar simulators, such as the PLATO Simulator (Marcos-Arenal et al., 2014). That will possibly require the engineering of new variables, but we will be able in principle to control the correlations between the predictors and the response.

The results indicate that the CS features are slightly more important for the model than the GWPS ones, although they are both comparable in terms of the quality assessment metrics. This is probably a consequence of the fact that the CS results from the product between the normalised GWPS and the ACF, thus amplifying peaks present in both of them and attenuating signals possibly resulting from instrumental effects, that manifest themselves differently in the GWPS and the ACF.

The most important TS variables are the activity proxies. They are relevant in the sense that they frequently make to the top-10 of most important variables, even when CS and GWPS features are present in the data set (recall figs. B.1 and B.2, for instance), and the performance of the model is still acceptable at a certain extent if predictions are carried out using uniquely this family of predictors. It becomes then apparent that it is relevant to extract all activity proxies possible (photometric, magnetic, and others) directly from the light curves and to use them as predictors when building a model.

When considered isolated, the weakest features for the prediction of stellar rotation periods are the astrophysical ones. Alone, they always yield models with poor predictive performance, useless for practical applications. However, the Flicker in Power values and the effective temperature sometimes reveal themselves within the most important variables, even in the presence of CS and GWPS features (recall figs. 4.8, B.2, B.3, B.5 and B.6).

Notably, in all data sets where they are present, variables belonging to the 55- and 20-day filter categories are the most relevant for the models, which is not unrelated to the fact that most of the stars in the data set have rotation periods between 10 and 40 d.

Considering the data sets from DS0 to DS8, when compared to the RFs models (recall table 4.2), XGBoost performed worse, as assessed by the mean error, the interval-accuracies, the RMSE, and the MAE. The only exception is the model created with DS6, whose predictive performance is slightly better than that of the corresponding RF. However, according to the adjusted-R^2, the

XGBoost models were able to explain better than the RFs ones the variability of the response in all data sets, except in DS8. The opposite is verified for the models created with DS9 and DS10, albeit the RF and XGBoost models keep being of comparable quality in those cases. It is worth noting that the RF models were obtained with little tuning of the hyperparameters, and that their quality could possibly have been improved further if other parameters had been optimised, such as the number of trees in the forest, the fraction of the original data set assigned to each tree, or the maximum depth of a tree. Moreover, in the data sets of reduced size, when compared to the XGBoost one, the RF models took much less time to train. So, depending on the level of precision needed, a RF model can be robust enough to to deliver good predictions. When all extra "percentage" is important, than models trained with XGBoost seem to be the best option.

The differences between the training and testing RMSE and MAE, although natural, might indicate some level of overfitting during the training process of the models built from the DS0 to DS10 data sets. However, these levels of overfitting do not seem to affect the quality performance of the models. The differences between RMSE and MAE vary among the models created with different data sets, indicating different levels of variation of the individual errors, but these differences are not expressive.

Prior to rotation period filtering, the models struggled to predict very short rotation periods, typically below a few days, and periods larger than 45 d. These discrepancies were evidenced in the scatter plots of the models trained with the DS0 to DS8 data sets, and are further highlighted in figs. B.14 to B.16, where golden lines represent the ground truth values, and blue and green lines correspond respectively to predictions performed on the training and testing sets. The differences between the reference and the predicted values are larger for longer rotation periods, typically above approximately 45 d, and very short rotation periods, below seven days. Therefore, cases corresponding to K and M stars with rotation periods below 7 d or above 45 d are not suitable to train a predictive model using Kepler data. This is essentially due to (*a*) Kepler's quarters of 45 days, (*b*) at least two full cycles are needed in order to get reliable observations, (*c*) the process of stitching two or more Kepler time series is not trivial and error free, and (*d*) signals from stars with rotation periods smaller than seven days may be mimicked by fluxes from close binaries or classic pulsators. After the filtering of the rotation periods, the predictive power of the RF and XGBoost models improved, as became clear from the analysis of the models learnt from DS9 and DS10. By restricting the predictors to the set of the 33 most important variables identified by the RF and XGBoost models created with DS0 to DS8, and by filtering in stars with rotation periods between 7 and 45 d, we were able to train computational cheap but solid models, with good predictive performance.

While being characterised by a remarkable goodness of fit, above 97 % in the XGBoost case, the final models introduced in section 4.3 still exhibit a few outliers with predictions off the real value by up to 200 %. In principle, increasing the number of training cases would help to circumvent this problem and to improve the predictive power of the models. Additionally, we could perform feature engineering on the original light curves, in an attempt to get a better set of predictors.

Lastly, our results are on par with the ones presented by Breton et al. (2021). Focusing on the 10 % interval-accuracy metric, the classifier created with B21 attained slightly better results, but we have to take into account the fact that our accuracy metric is an approximation to the counterpart used to assess the quality of a classification model. Moreover, we did not carried out any kind of visual inspection of the results nor altered them after testing the model. An important aspect to highlight is the fact that we applied the *hold-out method*, that is, we used an unseen-before testing set to perform the assessment of the quality of the model. This testing set was not part of the training of the model nor participated in the CV carried out to optimise the hyperparameters of the models. When assessing the quality of a model on a testing set, the predictive performance is typically more pessimistic than that obtained during the training with CV (Hastie, Tibshirani and J. Friedman, 2009; Torgo, 2011). The accuracies claimed by Breton et al. (2021) seem to be obtained with CV on the whole data set, and there is no reference in the paper to the hold-out method. Therefore, we would expect the performance of their classifier to be worse than the one they claim if the quality of the model were assessed on an unseen before testing set.

CHAPTER 5

Conclusions

For a construction to be useful and not mere waste of mental effort, for it to serve as a stepping-stone to heigher things, it must first of all possess a kind of unity enabling us to see something more than the juxtaposition of its elements.
— Henri Poincaré (1954-1912)

All models are wrong, but some are useful.
— George E. P. Box (1919-2013)

Contents

IN THIS THESIS, we have discussed the possibility of using machine learning (ML) approaches to determine rotation periods of stars. Obtaining accurate and precise measurements of stellar rotation periods for a great number of targets is essential for the study of the evolution of stars and of the Galaxy, as well as to characterise planetary systems, to understand stellar angular momentum transport, and to estimate the age of stars. Recently, attempts have been made to apply *artificial neural networks* (ANNs) to light curves, and classification *random forests* (RFs) to tabular data to perform such predictions. On the one hand, the former typically requires heavy computational resources; on the other hand, concerning the latter, we argue that (*a*) a classifier might not be the best approach to predict the rotation period, given that this is a continuous variable, and (*b*) using rotation periods as predictors might weaken the prediction performance of the model in the presence of unseen testing data, which do not contain rotation period values and have not been used to train the model.

5.1 Main Conclusions

As motivated in chapter 1, our main goal was settled on building robust and efficient ML models for the automatic prediction of stellar rotation periods of K and M stars from the *Kepler* catalogue. To this end, we applied the regression RFs and *extreme gradient boosting* (XGBoost) algorithms to eleven data sets built from the sets originally published by Santos et al. (2019) and Breton et al. (2021) and trained several models with them. In section 1.1, we divided the main goal into four research questions, which we will now revisit and answer in turn.

Research Question 1

Which independent variables evince the highest level of correlation with the target variable?

In chapter 3, section 3.1.2, we carried out a statistical analysis of the groups of input variables identified in page 36 (Prot, Astro, TS, GWPS, CS, and Wav). In particular, we measured the correlation within each group of predictors, and between them and the response variable, the rotation period, $p_{\rm rot}$. We found that:

- All Prot features, *i.e.*, rotation periods obtained from the *autocorrelation function* (ACF) of the light curve, from the *global wavelet power spectrum* (GWPS) analysis, and from the *composite spectrum* (CS) method, have strong positive correlations among them and with the response (recall fig. 3.4, left panel); we suspect that those predictors with correlations close to 1 were used to obtain the reference values of the rotation period published by Santos et al. (2019), since the ground truth was estimated by resorting on the ACF, GWPS, and CS methods.
- The unknown Wav variables exhibit no level of correlation among them and with the response (recall fig. 3.4, right panel).
- Out of the *time series* (TS) variables, the ones related to the photometric activity proxy, $S_{\rm ph}$, and its error, and to the control parameters, H and G, have a strong negative correlation with the target variable (recall fig. 3.5, left panel); variables related to the length of the light curve have small to no correlation with the response.
- Concerning the Astro predictors, only the logarithm of the surface gravity, $\log g$, is positively correlated with the rotation period; all other predictors are negatively correlated with the target variable, or have little to zero correlation with it; of those, the mass of the star, m, and the FliPer values for ν_C equal to 0.7 and 7 µHz, $f_{0.7}$ and f_7 respectively, have a non-negligible correlation with the response (recall fig. 3.5, right panel).
- In the CS features, we could identify three subgroups of variables: one exhibiting negative correlations with the response, one having positive correlations, and another (the largest) with little to no correlation with the target variable (recall fig. 3.6); from the former, the variables that stand out the most are related to the photometric activity proxy calculated on the CS, and to the period and amplitude of the first Gaussian function fit to the CS; from

the second group, the variables related to the central period and standard deviation of the Gaussian function fitted to the CS are the most prominent.

- Finally, in the GWPS group, similarly to the CS variables, we identify three subgroups, composed of variables having positive correlation with the response, negative correlation, and little to no correlation with the target variable (recall fig. 3.7); the first subgroup (the smallest of the three) is dominated with features related to the period of the first Gaussian function fitted to the GWPS; in the second subgroup, predictors concerning S_{ph} measured on the GWPS and its error, and to the number of Gaussian functions fitted to the GWPS stand out; we also identify variables related to the period of the fourth, fifth, and sixth Gaussian functions fitted to the GWPS as having non-negligible correlation with the response.

Research Question 2

How does a regression ML model trained on input variables classically used to estimate stellar rotation periods compare to the classifier developed by Breton et al. (2021)?

We addressed the problem of building supervised machine learning models to predict stellar rotation periods in chapter 3. We decided to use the RFs and XGBoost algorithms, following a regression approach. Given that Breton et al. (2021) built a classifier with their data set (hereafter B21), we developed a metric, the interval-based accuracy, that allowed us to convert predicted rotation periods into a proportion of successful events. We used this metric as a way to directly compare the results obtained with the two approaches (regression *vs*. classification).

In chapter 4, we carried out our experiments and built two or more models per data set based on RFs and XGBoost. Initially, we used all stars available in the base training set. Overall, the results show that the predictive performance of the models trained with RFs with a data set, DS0, equivalent to B21, are on par with the performance of the classifier trained with the latter, when we are willing to accept a 10 % error in the predictions. The predictive performance of the model trained with XGBoost is slightly worse than that of the previous model. However, the goodness of fit, as measured by the mean absolute value of the relative residuals, μ_{err}, and by the adjusted coefficient of determination, R^2_{adj}, indicates that both models are robust, able to explain most of the variability of the response (circa 95 %), and that they are wrong approximately 10 % of the time.

When we filter out stars with rotation periods below 7 d and above 45 d, the overall performance of the models increase considerably, with about 96 % to 98 % of the variability of the response explained by the predictors, and the 10 % interval-accuracy equal to approximately 90 %. In this case, the models were wrong no more than 2.3 % of the time.*

We note that we used the hold-out technique when training and testing our models and, hence, we assessed their performance on a testing set, containing unseen stars, that were not used during the training process. Breton et al. (2021) do not mention such a method, and apparently they used the whole data set to perform cross-validation (CV) during the training of their three classifiers

* We recall that the filtering was motivated by the fact that the original data was most possibly contaminated with classical pulsators and close binaries, which can modulate signals similar to rotation periods below typically 7 d, and with unreliable *Kepler* measurements of stars with rotation periods above 45 d.

based on RFs. In addition, we did not perform visual inspection nor adjustments of the results, as Breton et al. (2021) did. The results we report were obtained directly from the application of the models on the testing sets, without further processing.

Therefore, in view of the results we present here, we conclude that our approach is at least on par with with the classifier developed by Breton et al. (2021).

Research Question 3

Which sets of input variables are mandatory for obtaining a reliable ML model, with good predictive performance?

As mentioned in the answer to the research question 1, in chapter 3 we created several data sets, nine of which were based on the groups of predictors we identified in section 3.1. We used those data sets to train regression RFs and XGBoost models, and assess their quality using several metrics.

In view of the results reported in chapter 4, we conclude that the most important variables to train a reliable ML model with good predictive performance are the CS and GWPS ones. When a model is built with data devoid of these variables, its quality drops abruptly when compared to models built with data sets containing at least one of them. Moreover, the results from the models built with DS10 show that even when we only use ten predictors to train a model, we still get results comparable to Breton et al. (2021), provided that the central periods of the first Gaussian function fitted on the CS and on the GWPS are present in the data set.

Features of the TS type, *i.e.*, related to the structure of the light curve, such as S_{ph}, have some relevance for the estimation of stellar rotation periods, as well, since models built with CS, GWPS, and TS predictors are still able to explain about 88 to 91 % of the variability of the response. However, when used isolated, TS variables do not have the same level of importance (predictive power) as the other two.

Research Question 4

Is it possible to build an optimal subset of predictors from the set of available explanatory variables from which robust regression ML models, with good predictive performance, can be trained?

This question was addressed in section 4.3, where we used two new data sets of reduced size, DS9 and DS10. These sets were built by selecting the most important features, as measured during the training of the models obtained with the data sets DS0 to DS8, and by removing stars with rotation periods below 7 d and above 45 d, due to the characteristics of the *Kepler* space observatory and astrophysical constrains.

Our results show clearly that RFs and XGBoost models trained with these data sets of reduced size have comparable performance to B21, and to the best models we obtained before, with the DS0 and DS2 data sets. We claim a reduction of circa 150 to 170 predictors when compared to the

largest data set (DS0). The most important predictors, as measured by the **ranger** and **xgboost** R packages, are the central periods and standard deviations of the first Gaussian function fit to the CS and GWPS, followed by the photometric index proxy, $S_{\rm ph}$. Using a data set composed of only those variables allows one to train RF and XGBoost models with performances comparable to those built with much larger data sets. By reducing the size of the sets in this order of magnitude, we will be able to improve considerably the training time. With the resources we had access to, *i.e.*, shared machines on which several processes were running simultaneously, we were able to observe reductions in the training time of several hours.

Final remarks. In conclusion, our main goal has been achieved: we were able to build ML models to analyse *Kepler* data, and that allow us to automatically and reliably predict the rotation periods of K and M stars. Our results show that the predictive performance of the models depends not so much on the number of input variables, but on their type.

As a remark, we believe that rotation periods calculated with classical methods should not be used as predictors, because, on the one hand, they already contain the values we are aiming to obtain and, on the other hand, they are decreasing the predictive performance of the model when applying it to unseen stars, whose rotation periods we want to predict. Moreover, we suggest that feature engineering should be performed on synthetic light curves, built from robust stellar simulators, in order to prevent any bias introduced to the model by the methods used to estimate the reference values.

5.2 Prospects and Future Work

The methodology presented in this thesis can be improved in several ways. The open questions that follow from this work are:

- How much correlated features affect the predictive performance of the models? In particular, would we obtain the same set of the most important predictors, specially in the case of RFs, if feature selection would have been carried out prior to model training, removing correlated variables?
- Are there confounding variables in the base data set? How do they affect the predictive performance of the models?
- Would we be able to improve the quality of the RF models if we had increased the number of trees during training?
- How does the predictive performance of the models increase if more data is made available?
- Do the large outliers, with values approximately equal to the double of the reference ones, correspond to harmonics the models might be capturing?
- Is it possible to engineer new features from the light curves in order to improve the performance of the models?
- How do models trained from synthetic data perform comparatively to models created with real data?

These questions lead us to the following potentially interesting research directions.

Feature selection prior to model training. Feature selection should be performed on the base data set, by (*a*) investigating correlations within predictors and between those and the target variable, and (*b*) by investigating the presence of confounding variables. The task related to the first topic would be similar to the one carried out in section 3.1.2, but with the extra step of eliminating colinear features before model training. Given that most of them are derived variables, a possible criterion to select surviving features could be the ease with which they are obtained. Confounding variables are known to introduce spurious association between the independent and the target variables. Their presence in the original data set should be examined. Similar models to the ones presented in this work should be learnt form the data set after feature selection, and results compared.

Optimise the number of trees in RFs. We did not vary the number of trees while training the models based on RFs, due to computational resources constrains. An interesting topic could be to investigate how much increasing that parameter affects the predictive performance of the models.

Investigate the presence of outliers. Some predicted stellar rotation periods are approximately equal to the double of their reference values. This seems to indicate that the models are capturing harmonics, and this is worth further investigation. In particular, increasing the number of observations in the data set might reduce the number of outliers produced by the models.

Perform feature engineering. We can use the KEPSEISMIC light curves to perform time series feature engineering. That will require further research on how to extract important features from the light curves. A good starting point is the work by Cerqueira, Moniz and Soares (2021).

Using synthetic data. Having identified a set of optimal features to predict stellar rotation periods, it is relevant to train models using tabular data extracted from robust stellar simulators, such as the PLATO Simulator (Marcos-Arenal et al., 2014), and assess the performance of such models. On the one hand, we can build a training data set with an uniform distribution for the ground truth; on the other hand, since we are not introducing bias due to the presence of predictors that were used to estimate the stellar rotation periods used as reference, we are expecting those models to be reliable. An additional advantage is that we can train the models on stars with any possible rotation period, without being concerned with spurious features, misleading signals introduced by classical pulsators or close binaries, and unreliable data due to poor stitching of light curves. These models can be used to update the rotation period catalogues currently available.

Using ANNs. *There is no free lunch in ML*: there is no method that dominates all others over all possible data sets. While other methods could be applied to some of the data sets created during this project, mainly DS2, DS6, DS9, and DS10, to decide which learner produces the best results, we are not expecting to get improvements by using linear regression, generalised additive models,

lasso, ridge, or support vector machines. We can follow a different type of research line, and estimate stellar rotation periods from a large number of pre-processed light curves, by applying *deep learning* (DL) methods, such as *convolutional neural networks*. Data can be obtained from synthetic and knitted real light curves. Perhaps the predictive performance can be improved with this approach, but these are computational demanding methods and, hence, the effort needed to train the models might not compensate for the gain.

Bibliography

Aerts, Conny, Stephane Mathis and Tamara Rogers (Sept. 2019). 'Angular Momentum Transport in Stellar Interiors'. In: pp. 35–78. URL: http://arxiv.org/pdf/1809.07779v1.

Agresti, Alan, Christine Franklin and Bernhard Klingenberg (2018). Global Edition. Pearson, p. 817.

Angus, Ruth, Angus Beane et al. (2020). 'Exploring the evolution of stellar rotation using Galactic kinematics'. In: *The Astronomical Journal* 160.2, p. 90.

Angus, Ruth, Timothy Morton et al. (2018). 'Inferring probabilistic stellar rotation periods using Gaussian processes'. In: *Monthly Notices of the Royal Astronomical Society* 474.2, pp. 2094–2108.

Archdeacon, T.J. (1994). *Correlation and Regression Analysis: A Historian's Guide*. University of Wisconsin Press. ISBN: 9780299136543. URL: https://books.google.pt/books?id=QmFFpQW8V8EC.

Baron, Michael (2019). *Probability and statistics for computer scientists*. Chapman and Hall/CRC.

Benbakoura, M. et al. (Nov. 2019). 'Evolution of star-planet systems under magnetic braking and tidal interaction'. In: URL: http://arxiv.org/pdf/1811.06354v1.

Biehl, Michael et al. (2018). 'Machine learning and data analysis in astroinformatics.' In: *ESANN*.

Bishop, Christopher M et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Blancato, Kirsten et al. (2022). 'Data-driven Derivation of Stellar Properties from Photometric Time Series Data Using Convolutional Neural Networks'. In: *The Astrophysical Journal* 933.2, p. 241.

Borucki, William et al. (Feb. 2009). 'KEPLER: Search for Earth-Size Planets in the Habitable Zone'. In: *Transiting Planets*. Ed. by Frédéric Pont, Dimitar Sasselov and Matthew J. Holman. Vol. 253, pp. 289–299.

Borucki, William J et al. (2010). 'Kepler planet-detection mission: introduction and first results'. In: *Science* 327.5968, pp. 977–980.

Breiman, Leo (Aug. 1996a). 'Bagging predictors'. In: *Machine Learning* 24.2, pp. 123–140. ISSN: 1573-0565.

— (1996b). *Bias, variance, and arcing classifiers*. Tech. rep. 460. Statistics Department, University of California at Berkeley.

— (2001). 'Random forests'. In: *Machine learning* 45.1, pp. 5–32.

Breton, Sylvain N et al. (2021). 'ROOSTER: a machine-learning analysis tool for Kepler stellar rotation periods'. In: *Astronomy & Astrophysics* 647, A125.

Brun, Allan Sacha and Matthew K. Browning (2017). 'Magnetism, dynamo action and the solar-stellar connection'. In: *Living Reviews in Solar Physics* 14.1, pp. 1–133.

Bugnet, L et al. (2018). 'FliPer: A global measure of power density to estimate surface gravities of main-sequence solar-like stars and red giants'. In: *Astronomy & Astrophysics* 620, A38.

Callahan, James J (2010). *Advanced calculus: a geometric view*. Vol. 1. Springer.

Carroll, Bradley W. and Dale A. Ostlie (2017). *An Introduction to Modern Astrophysics*. Ed. by San Francisco: Pearson Addison-Wesley. 2nd (International). URL: https://www.bibsonomy.org/bibtex/2cd2d46d88b1408585d589f96be3ddeaa/ad4.

Casella, George and Roger L Berger (2001). *Statistical inference*. Cengage Learning.

Ceillier, T. et al. (July 2017). 'Surface rotation of Kepler red giant stars'. In.

Cerqueira, Vitor, Nuno Moniz and Carlos Soares (Apr. 2021). *VEST: Automatic Feature Engineering for Forecasting*. URL: http://arxiv.org/abs/2010.07137v1;%20http://arxiv.org/pdf/2010.07137v1.

Chen, Tianqi, Michaël Benesty and Tong He (2018). *Understand Your Dataset with Xgboost*. URL. cran.r-project.org/web/packages/xgboost/vignettes/discoverYourData.html, Accessed on 2022-09-18.

Chen, Tianqi and Carlos Guestrin (June 2016). *XGBoost: A Scalable Tree Boosting System*.

Dobson, Annette J and Adrian G Barnett (2018). *An introduction to generalized linear models*. Chapman and Hall/CRC.

Efron, Bradley and Robert Tibshirani (1986). 'Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy'. In: *Statistical science*, pp. 54–75.

Eggenberger, P. et al. (Nov. 2009). *Effects of rotation on the evolution and asteroseismic properties of red giants*. URL: http://arxiv.org/pdf/0911.5307v1.

Encyclopaedia Britannica, The Editors of (June 2022). *Joseph von Fraunhofer. Encyclopedia Britannica*. URL: www.britannica.com/biography/Joseph-von-Fraunhofer (visited on 14/09/2022).

Freund, Yoav (1995). 'Boosting a weak learning algorithm by majority'. In: *Information and computation* 121.2, pp. 256–285.

Freund, Yoav and Robert E Schapire (1997). 'A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting.' In: *J. Comput. Syst. Sci.* 55.1, pp. 119–139.

— (1999). 'Adaptive game playing using multiplicative weights'. In: *Games and Economic Behavior* 29.1-2, pp. 79–103.

Friedman, Jerome H (2001). 'Greedy function approximation: a gradient boosting machine'. In: *Annals of statistics*, pp. 1189–1232.

García, RA et al. (2014). 'Rotation and magnetism of Kepler pulsating solar-like stars-Towards asteroseismically calibrated age-rotation relations'. In: *Astronomy & Astrophysics* 572, A34.

Géron, Aurélien (2017). 'Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems'. In: *Sebastopol, CA: O*.

Gomes, Nuno R. C. (2016). 'Imaging with the VLTI'. PhD thesis. Porto, Portugal: University of Porto, p. 340.

Hasinoff, Samuel W (2014). 'Photon, Poisson Noise'. In: *Computer Vision, A Reference Guide* 4.

Hastie, Trevor, Robert Tibshirani and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second. Springer.

Haykin, Simon (2009). *Neural Networks and Leraning Machines*. PHI Learning, p. 944.

Hosmer, David W. and Stanley Lemeshow (2013). 3rd ed. Wiley-Interscience Publication. ISBN: 978-0-470-58247-3.

Howell, Steve B et al. (2014). 'The K2 mission: characterization and early results'. In: *Publications of the Astronomical Society of the Pacific* 126.938, p. 398.

Huber, Daniel et al. (May 2016). *The K2 Ecliptic Plane Input Catalog (EPIC) and Stellar Classifications of 138,600 Targets in Campaigns 1-8*. URL: http://arxiv.org/pdf/1512.02643v4.

Hyndman, Rob J and Anne B Koehler (2006). 'Another look at measures of forecast accuracy'. In: *International journal of forecasting* 22.4, pp. 679–688.

James, Gareth et al. (2013). *An Introduction to Statistical Learning – with Applications in R*. Vol. 103. Springer Texts in Statistics. New York: Springer. ISBN: 978-1-4614-7137-0.

Karttunen, Hannu et al. (2007). *Fundamental astronomy*. Springer.

Kuhn, Max, Kjell Johnson et al. (2013). *Applied predictive modeling*. Vol. 26. Springer.

Kvålseth, Tarald O (1985). 'Cautionary note about R 2'. In: *The American Statistician* 39.4, pp. 279–285.

Laurae (2016). *xgboost: "Hi I'm Gamma. What can I do for you?" — and the tuning of regularization*. https://medium.com/data-design/xgboost-hi-im-gamma-what-can-i-do-for-you-and-the-tuning-of-regularization-a42ea17e6ab6, Accessed on 2022-09-20.

Liu, Bing (2011). 2nd ed. Springer Heidelberg Dordrecht London New York. ISBN: 978-3-642-19459-7.

Marcos-Arenal, P et al. (2014). 'The PLATO Simulator: modelling of high-precision high-cadence space-based imaging'. In: *Astronomy & Astrophysics* 566, A92.

Mathur, Savita, RA García et al. (2014). 'Magnetic activity of F stars observed by Kepler'. In: *Astronomy & Astrophysics* 562, A124.

Mathur, Savita, Daniel Huber et al. (2017). 'Revised stellar properties of Kepler targets for the Q1-17 (DR25) transit detection run'. In: *The Astrophysical Journal Supplement Series* 229.2, p. 30.

McQuillan, Amy, Tsevi Mazeh and Suzanne Aigrain (2014). 'Rotation periods of 34,030 Kepler main-sequence stars: the full autocorrelation sample'. In: *The Astrophysical Journal Supplement Series* 211.2, p. 24.

Miglio, A. et al. (Feb. 2013). 'Galactic archaeology: mapping and dating stellar populations with asteroseismology of red-giant stars'. In: *Monthly Notices of the Royal Astronomical Society* 429.1, pp. 423–428. ISSN: 1365-2966.

Mount, John and Nina Zumel (2020). *Using vtreat with Regression Problems*. https://github.com/WinVector/vtreat/blob/master/Examples/Regression/Regression_FP.md, Accessed on 2022-09-25.

Mullally, S (2020). *MAST Kepler Archive Manual*.

Murphy, K.P. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning. MIT Press. ISBN: 9780262018029. URL: https://books.google.co.in/books?id=NZP6AQAAQBAJ.

Nolan, Deborah and Terry P Speed (2001). *Stat labs: Mathematical statistics through applications*. Springer Science & Business Media.

Pichara Baksai, Karim Elías, Pavlos Protopapas and D Leon (2016). 'Meta-classification for variable stars'. In.

Pontius, Robert Gilmore, Olufunmilayo Thontteh and Hao Chen (2008). 'Components of information for multiple resolution comparison between maps that share a real variable'. In: *Environmental and ecological statistics* 15.2, pp. 111–142.

Ramachandran, Kandethody M and Chris P Tsokos (2020). *Mathematical statistics with applications in R*. Academic Press.

Raschka, Sebastian and Vahid Mirjalili (Sept. 2017). *Python Machine Learning*. Second. Livery Place 35 Livery Street Birmingham B3 2PB, UK: Packt Publishing Ltd.

Reinhold, Timo, Ansgar Reiners and Gibor Basri (2013). 'Rotation and differential rotation of active Kepler stars'. In: *Astronomy & Astrophysics* 560, A4.

Ricker, George R. et al. (Oct. 2015). 'The Transiting Exoplanet Survey Satellite'. In.

Ripley, Brian D (2007). *Pattern recognition and neural networks*. Cambridge university press.

Santos, A. R. G. et al. (Aug. 2019). *Surface rotation and photometric activity for Kepler targets I. M and K main-sequence stars*.

Schapire, Robert E (1990). 'The strength of weak learnability'. In: *Machine learning* 5.2, pp. 197–227.

— (2003). 'The boosting approach to machine learning: An overview'. In: *Nonlinear estimation and classification*, pp. 149–171.

Shapiro, AI et al. (2020). 'Inflection point in the power spectrum of stellar brightness variations-I. The model'. In: *Astronomy & Astrophysics* 633, A32.

Skumanich, Andrew (1972). 'Time scales for CA II emission decay, rotational braking, and lithium depletion'. In: *The Astrophysical Journal* 171, p. 565.

Stahler, Steven W. and Francesco Palla (2008). Wiley. ISBN: 9783527618682. URL: https://books.google.ca/books?id=X91UBLr64FMC.

Strassmeier, K. G. (2009). 'Starspots'. In: *Astronomy and Astrophysics Review* 17.3, pp. 251–308.

Strobl, Carolin et al. (2007). 'Bias in random forest variable importance measures: Illustrations, sources and a solution'. In: *BMC bioinformatics* 8.1, pp. 1–21.

Titterington, Michael (2010). 'Neural networks'. In: *WIREs Computational Statistics* 2.1, pp. 1–8.

Torgo, Luís (2011). *Data mining with R: learning with case studies*. chapman and hall/CRC.

Usselman, M. C. (Aug. 2022). *William Hyde Wollaston. Encyclopedia Britannica*. URL: www.britannica.com/biography/William-Hyde-Wollaston (visited on 14/09/2022).

Van Saders, Jennifer L et al. (2016). 'Weakened magnetic braking as the origin of anomalously rapid rotation in old field stars'. In: *Nature* 529.7585, pp. 181–184.

Vapnik, Vladimir N. (2000). *The nature of statistical learning theory*. Second. Statistics for Engineering and Information Science. Springer-Verlag, New York, pp. xx+314. ISBN: 0-387-98780-0.

Wright, Marvin N, Stefan Wager and Philipp Probst (2019). 'A fast implementation of random forests'. In: *R package version 0.11* 2, pp. 123–136.

Wright, Marvin N and Andreas Ziegler (2017). 'ranger: A fast implementation of random forests for high dimensional data in C++ and R'. In: *Journal of Statistical Software*.

Zumel, Nina and John Mount (2016). 'vtreat: a data.frame Processor for Predictive Modeling'. In: *arXiv preprint arXiv:1611.09477*.

— (2019). *Practical data science with R*. Simon and Schuster.

Appendices

APPENDIX A

Tables

Table A.1: Names of the 180 predictors, as they appear in the data set, and their description. Indicated are the group to which they belong, and the effective number of variables associated to each of them (N). *Astrophysical quantities* are labelled "astro"; "ts" stands for *time series*, "cs" for *composite spectrum*, "gwps" for *global wavelet power spectrum*, and "wav" is an unknown quantity; "prot" corresponds to rotation periods obtained using ACF, CS, and time-period analysis; $xx = 20, 55, 80$.

Variable	Description	Group	Observations	N
`teff`	$T_{\rm eff}$, effective temperature of the star	astro		1
`teff_eup`, `teff_elo`	upper and lower errors for $T_{\rm eff}$	astro		2
`logg`	$\log g$, logarithm of the surface gravity	astro		1
`logg_eup`, `logg_elo`	upper and lower errors of $\log g$	astro		2
`m`	M, the mass of the star, in solar masses	astro		1
`m_eup`, `m_elo`	upper and lower errors for M	astro		2
`f_07`, `f_7`, `f_20`, `f_50`	FliPer values for $\nu_{\rm C} = 0.7,\ 7,\ 20$ and $50\,\mu$Hz	astro		4
`kepmag`	*Kepler* magnitude from the *Kepler* input catalogue	astro	parameter linked to the quality of the stellar target	1
`length`	length of the light curve, in days	ts	parameter linked to the quality of the acquired light curve	1

Continued on next page

Table A.1: Names of the 180 predictors, as they appear in the data set, and their description. Indicated are the group to which they belong, and the effective number of variables associated to each of them (N). *Astrophysical quantities* are labelled "astro"; "ts" stands for *time series*, "cs" for *composite spectrum*, "gwps" for *global wavelet power spectrum*, and "wav" is an unknown quantity; "prot" corresponds to rotation periods obtained using ACF, CS, and time-period analysis; $xx = 20, 55, 80$. (Continued)

`bad_q_flag`	bad quarter flag	ts	parameter linked to the quality of the acquired light curve	1
`n_bad_q`	number of bad quarters in the light curve	ts	parameter linked to the quality of the acquired light curve	1
`start_time`, `end_time`	starting and ending time of the light curve	ts	parameter linked to the quality of the acquired light curve	2
`sph_acf_xx`, `sph_acf_err_xx`	$S_{\rm ph}$, photometric activity proxy, and its error, computed on the ACF for the *xx*-day filter	ts	mean of standard deviations over light curve segments of $5 \times P_{\rm rot}^{\rm ACF}$	6
`sph`, `sph_e`	$S_{\rm ph}$, photometric activity proxy	ts	it is a magnetic activity proxy	2
`g_acf_xx`	height of $P_{\rm ACF}$ for the *xx*-day filter	ts	$P_{\rm ACF}$ is the period of the highest peak in the ACF at a lag greater than zero; $G_{\rm ACF}$ is a control parameter	3
`h_acf_xx`	mean difference between height of $P_{\rm ACF}$ and the two local minima on both its sides for the *xx*-day filter	ts	$H_{\rm ACF}$ is a control parameter	3
`cs_noise_xx`	mean level of noise of the Gaussian functions fitted to CS for the *xx*-day filter	cs	this is the period uncertainty; the level of noise corresponds to the HWHM	3
`cs_chiq_xx`	χ^2 of the fit of the Gaussian function on CS for the *xx*-day filter	cs		3
`cs_n_fit_xx`	number of Gaussian function fitted to each CS for the *xx*-day filter	cs		3
`h_cs_xx`	amplitude of $P_{\rm CS}$ for the *xx*-day filter	cs	$P_{\rm CS}$ is obtained as the period of the fitted Gaussian of highest amplitude; it is a control parameter	3
`sph_cs_xx`, `sph_cs_err_xx`	$S_{\rm ph}$ and its error computed on CS for the *xx*-day filter	cs	mean standard deviations over light curve segments of $5 \times P_{\rm rot}^{\rm CS}$	6

Continued on next page

Table A.1: Names of the 180 predictors, as they appear in the data set, and their description. Indicated are the group to which they belong, and the effective number of variables associated to each of them (N). *Astrophysical quantities* are labelled "astro"; "ts" stands for *time series*, "cs" for *composite spectrum*, "gwps" for *global wavelet power spectrum*, and "wav" is an unknown quantity; "prot" corresponds to rotation periods obtained using ACF, CS, and time-period analysis; $xx = 20, 55, 80$.(Continued)

`cs_gauss_i_j_xx`	amplitude ($i = 1$), central period ($i = 2$), and standard deviation ($i = 3$) of the j^{th} ($j = 1, 2, \ldots 5$) Gaussian fitted in the CS with the *xx*-day filter	cs	CS is the product between the normalised GWPS and the ACF	45
`gwps_noise_xx`	mean level of noise of the Gaussian functions fitted to GWPS for the *xx*-day filter	gwps	this is the period uncertainty; the level of noise is given by HWHM	3
`gwps_chiq_xx`	χ^2 of the fit of the Gaussian function on GWPS for the *xx*-day filter	gwps		3
`gwps_n_fit_xx`	number of Gaussian functions fitted to each GWPS for the *xx*-day filter	gwps		3
`gwps_gauss_i_j_xx`	amplitude ($i = 1$), central period ($i = 2$), and standard deviation ($i = 3$) of the j^{th} ($j = 1, 2, \ldots 6$) Gaussian fitted in GWPS with the *xx*-day filter	gwps		54
`sph_gwps_xx`, `sph_gwps_err_xx`	S_{ph} and its error computed on GWPS for the *xx*-day filter	gwps	mean standard deviation over light curve segments of $5 \times P_{\text{rot}}^{\text{GWPS}}$	6
`wav_scl_max_xx`, `wav_scl_min_xx`	maximum and minimum of unknown quantity for the *xx*-day filter	wav		6
`prot_acf_xx`	rotation period extracted from the ACF analysis for the *xx*-day filter	prot		3
`prot_cs_xx`	rotation period extracted from the CS analysis for the *xx*-day filter	prot		3
`prot_gwps_xx`	rotation period extracted from the GWPS analysis for the *xx*-day filter	prot		3

APPENDIX B

Plots

In this chapter, we present all plots and figures referenced in the manuscript that are not included in the main text. The figures displayed in the following concern about variable importance, detailed scatter plots between the ground truth and the predicted rotation periods, and their marginal distributions.

B.1 Variable Importance

Random Forests

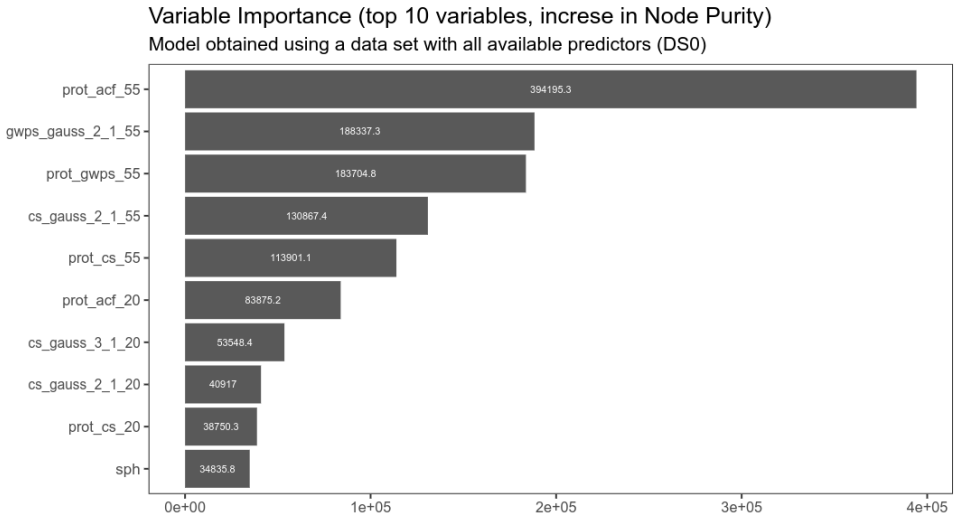

Figure B.1: Variable importance in the random forest model created with DS0.

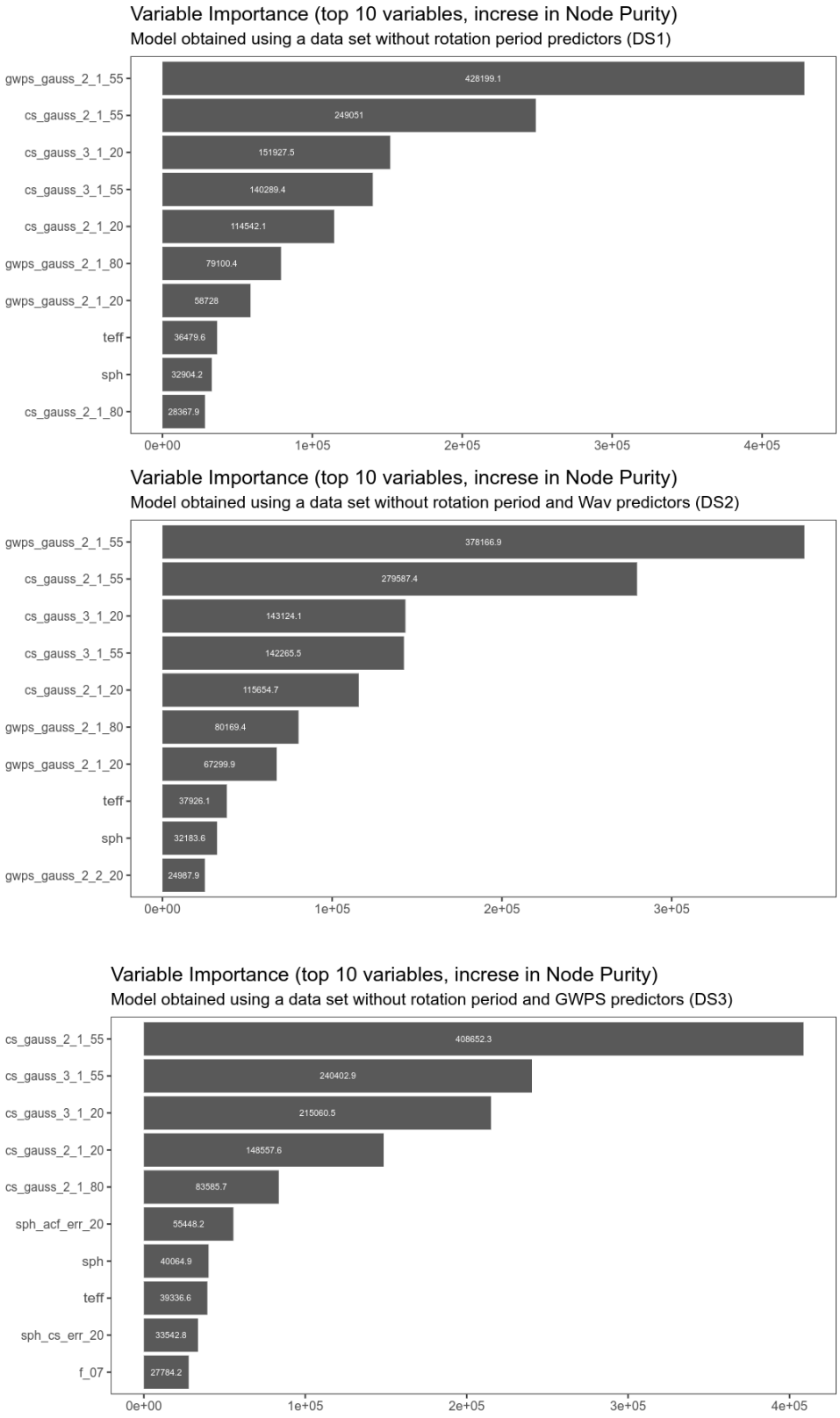

Figure B.2: Similar to fig. B.1, but for DS1 (top), DS2 (middle), and DS3 (bottom) models.

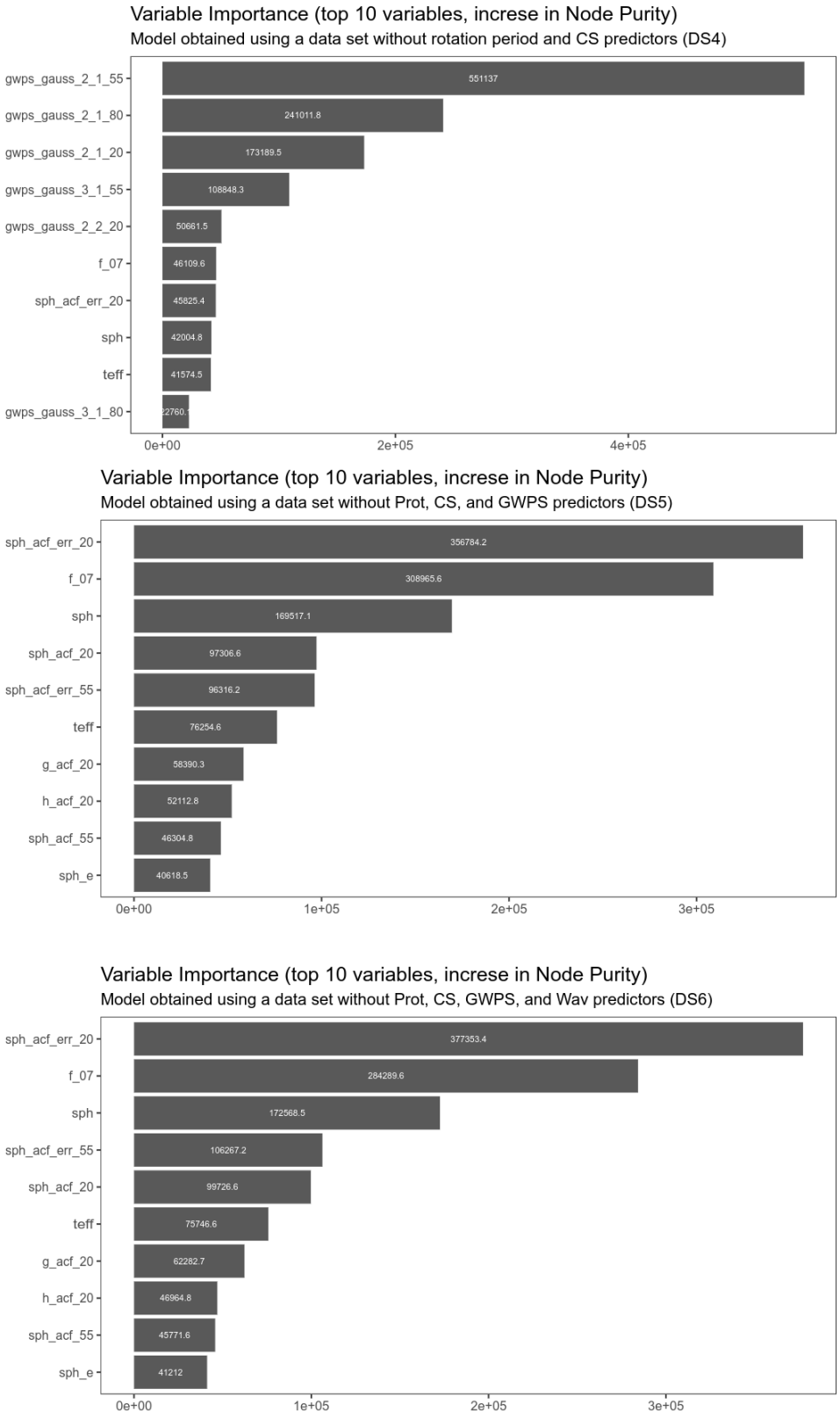

Figure B.3: Similar to figs. B.1 and B.2, but for DS4 (top), DS5 (middle), and DS6 (bottom) models.

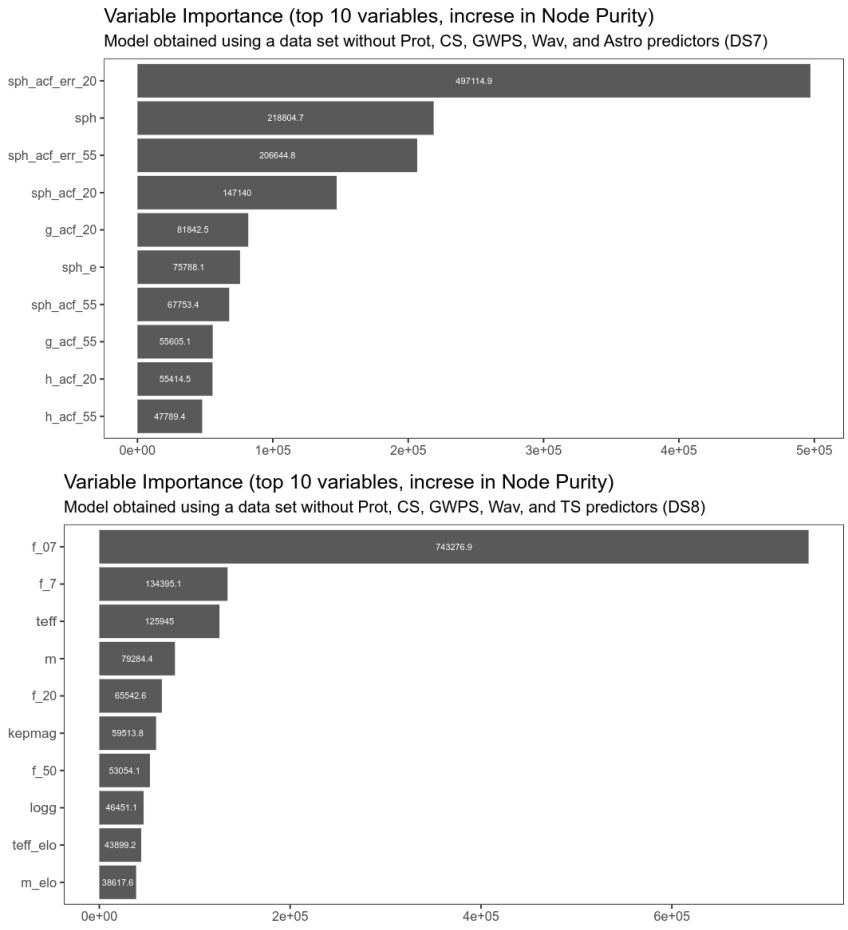

Figure B.4: Similar to figs. B.1 to B.3, but for DS7 (top), and DS8 (bottom) models.

XGBoost

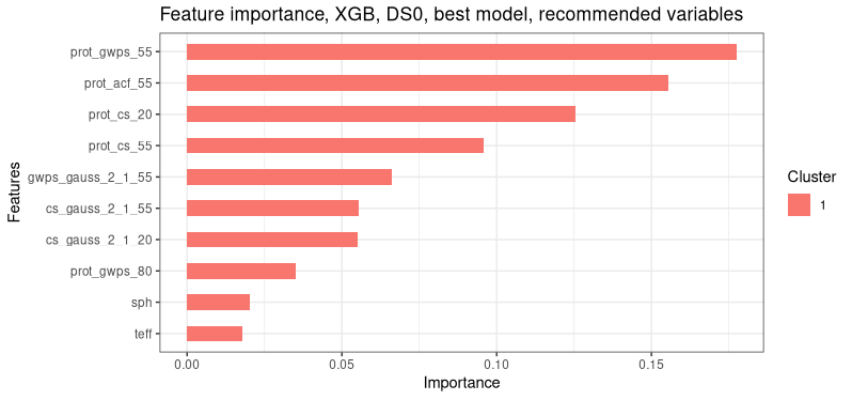

Figure B.5: Variable importance in the XGBoost model created with DS0.

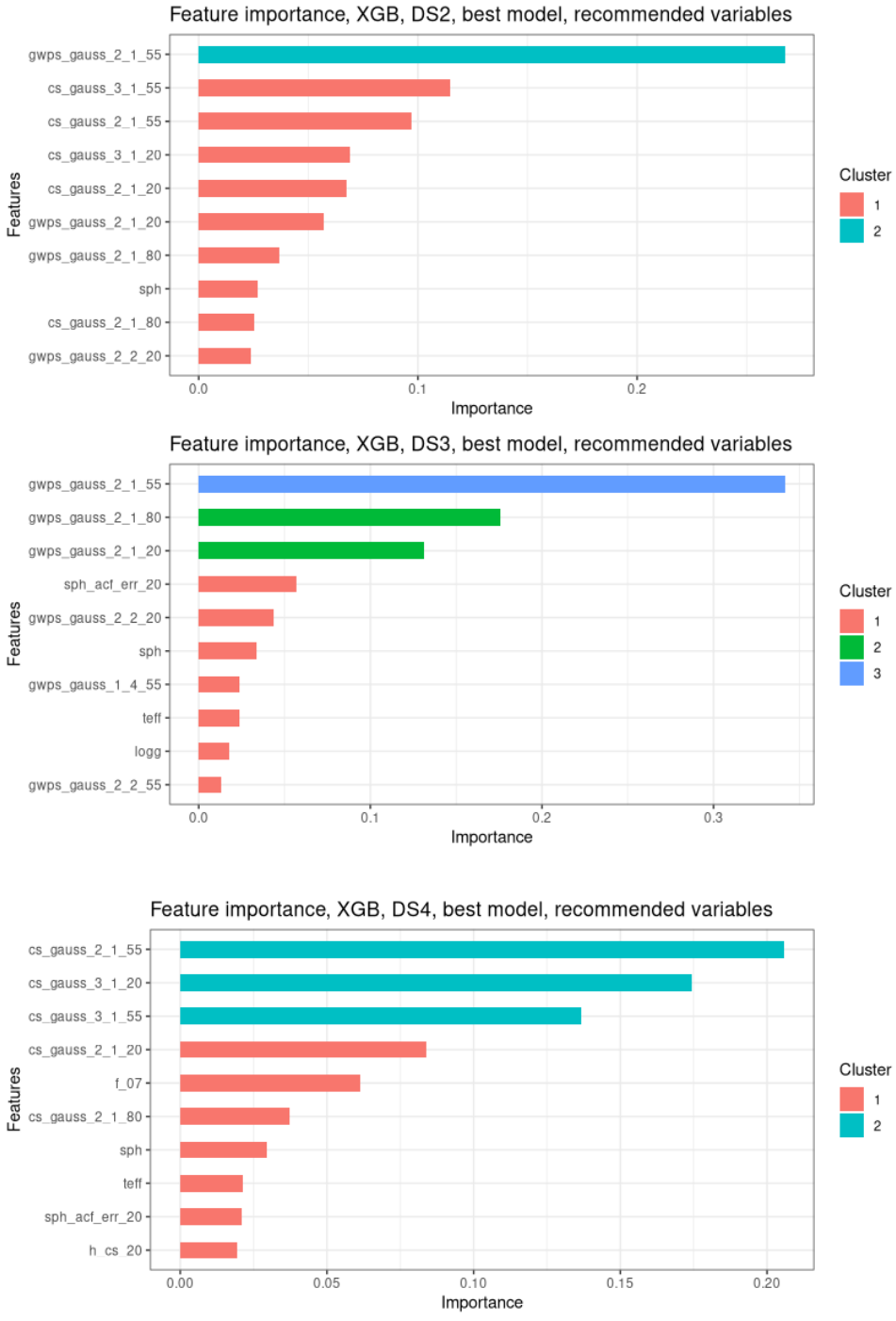

Figure B.6: Similar to fig. B.5, but for DS2 (top), DS3 (middle), and DS4 (bottom) models. The importance values were clustered, so that features with the same colour have similar importances.

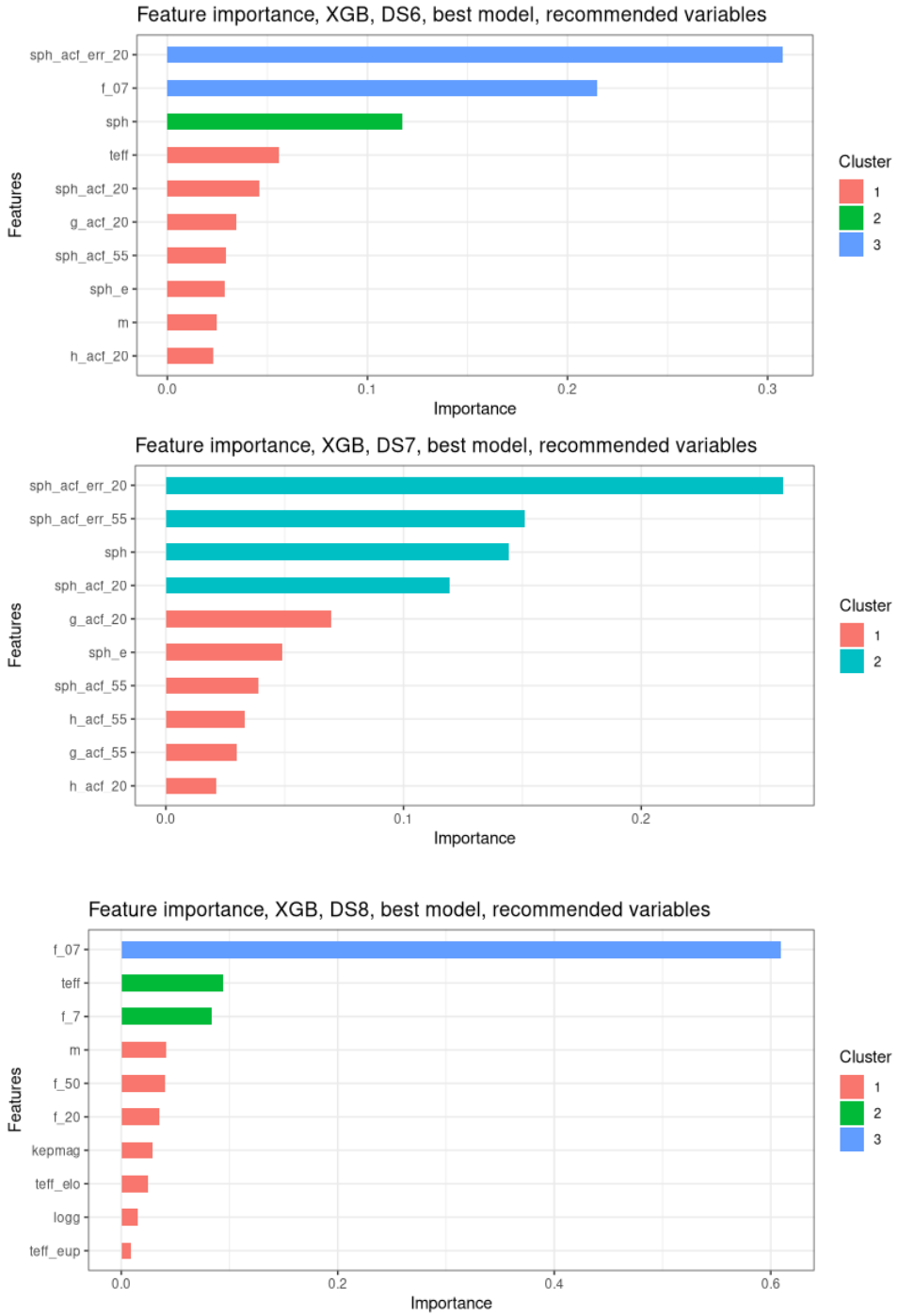

Figure B.7: Similar to figs. B.5 and B.6, but for DS6 (top), DS7 (middle), and DS8 (bottom) models.

B.2 Ground Truth *vs.* Predictions

Random Forests

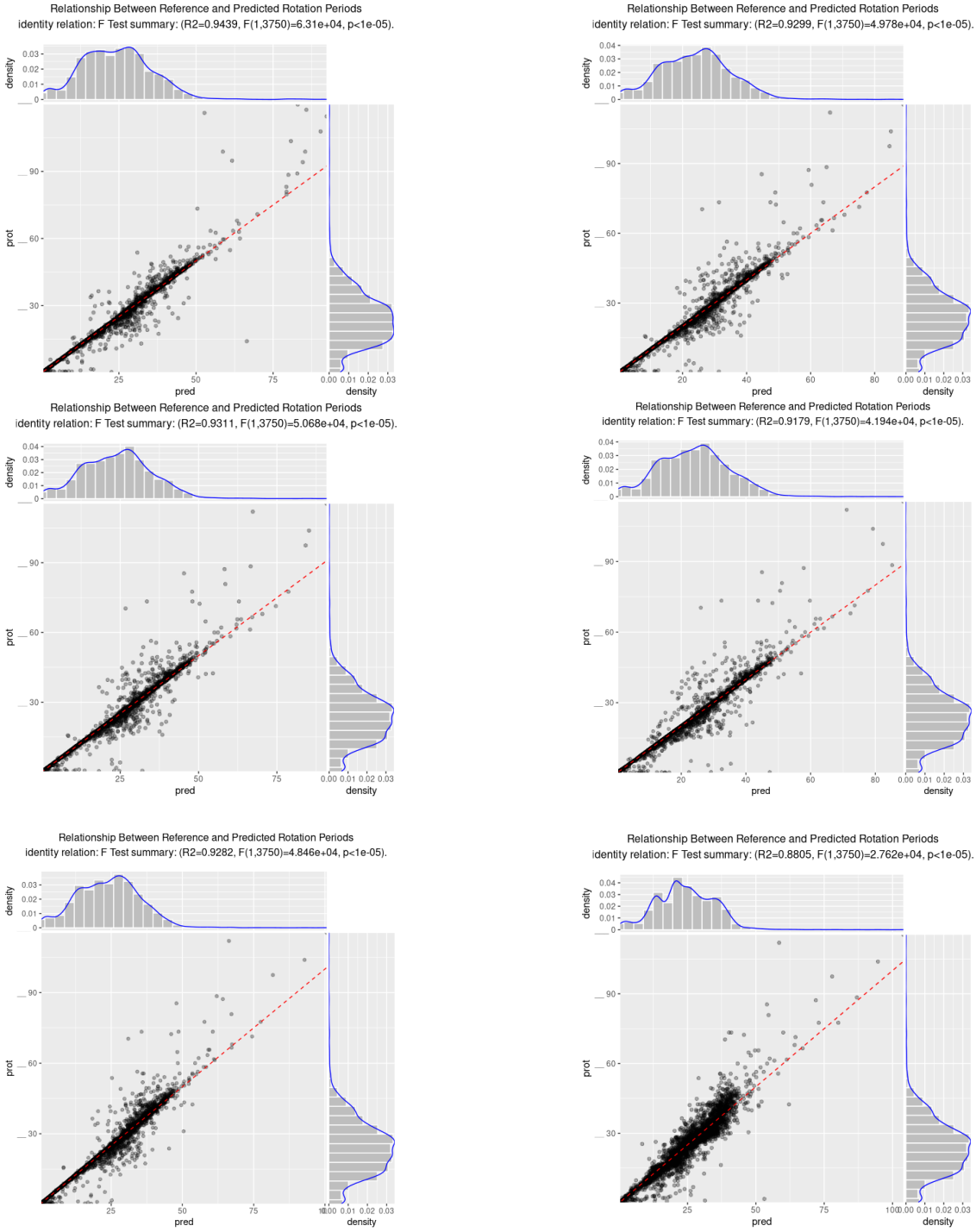

Figure B.8: Real values *vs.* predictions on the testing set for RF models built upon DS0 (top left), DS1 (top right), DS2 (middle left), DS3 (middle right), DS4 (bottom left), and DS5 (bottom right). The red dashed lines indicate the identity function.

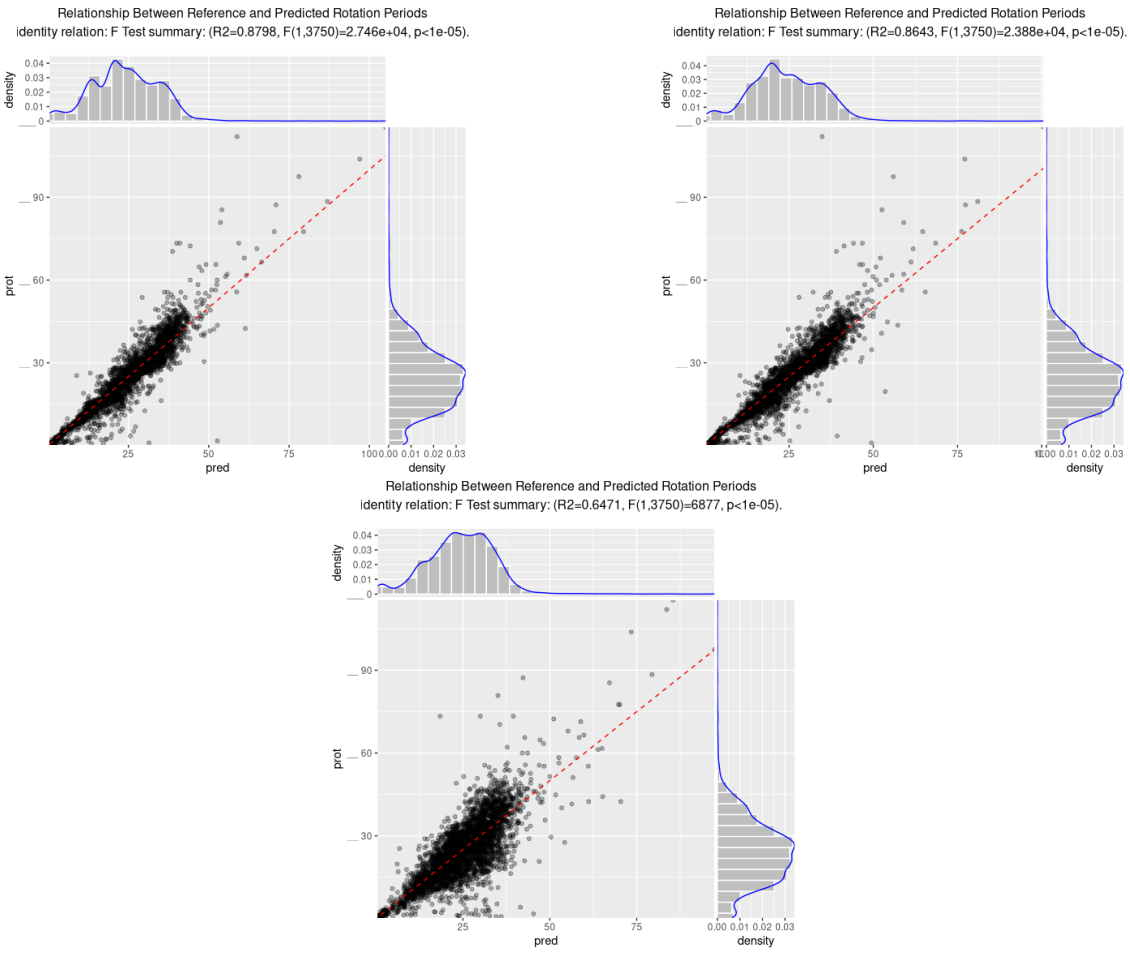

Figure B.9: Similar to fig. B.8, but for DS6 (top left panel), DS7 (top right panel), and DS8 (bottom panel).

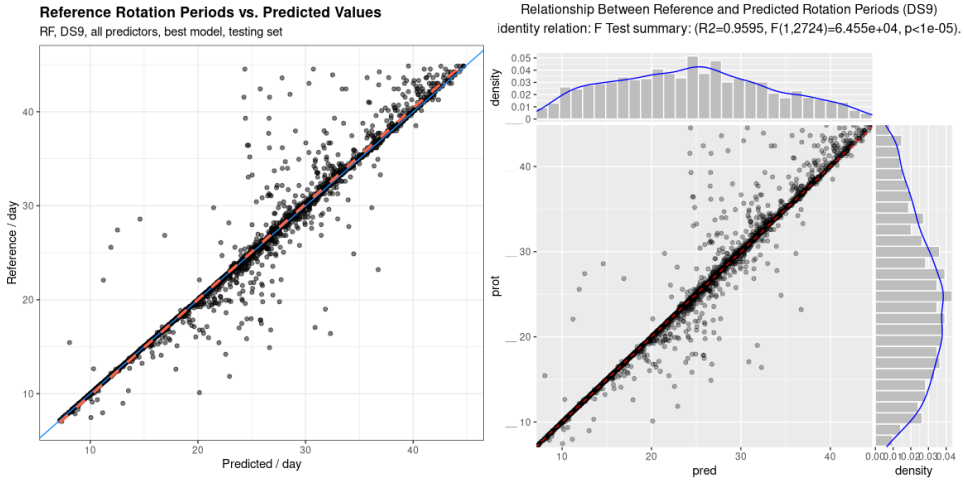

Figure B.10: Scatter plot of the reference rotation periods *vs.* the predictions for the RF model trained with the DS9 data set using all available variables. On the left panel, the blue solid line represents the identity function, and the red dashed line the linear model between the predicted and the true values; the right panel presents the identity function (red dashed line), and marginal histograms and density plots of the sample of predicted and reference rotation periods.

XGBoost

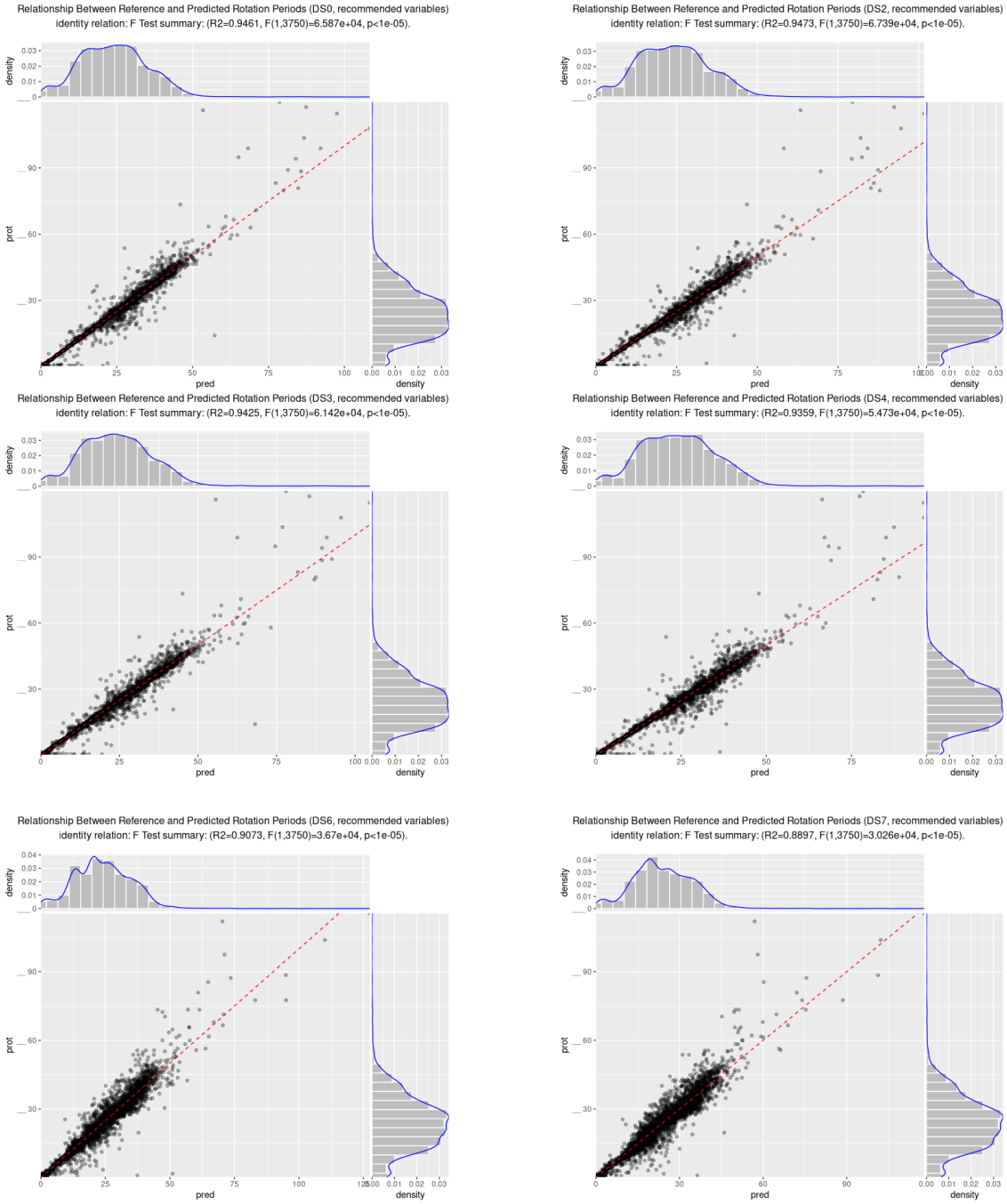

Figure B.11: Reference *vs.* prediction values on the testing sets for the XGBoost models built upon DS0 (top left), DS2 (top right), DS3 (middle left), DS4 (middle right), DS6 (bottom left), and DS7 (bottom right). The red dashed lines indicate the identity function.

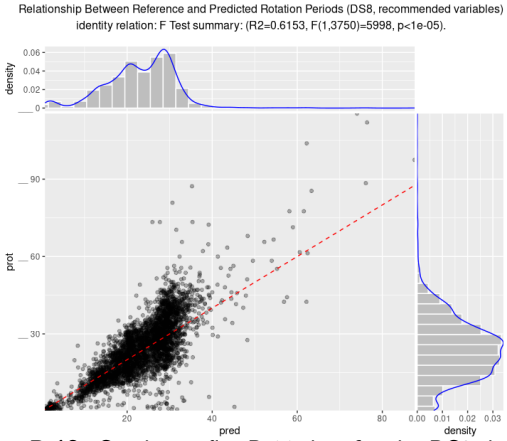

Figure B.12: Similar to fig. B.11, but for the DS8 data set.

B.3 Residuals and Error Metrics

Random Forests

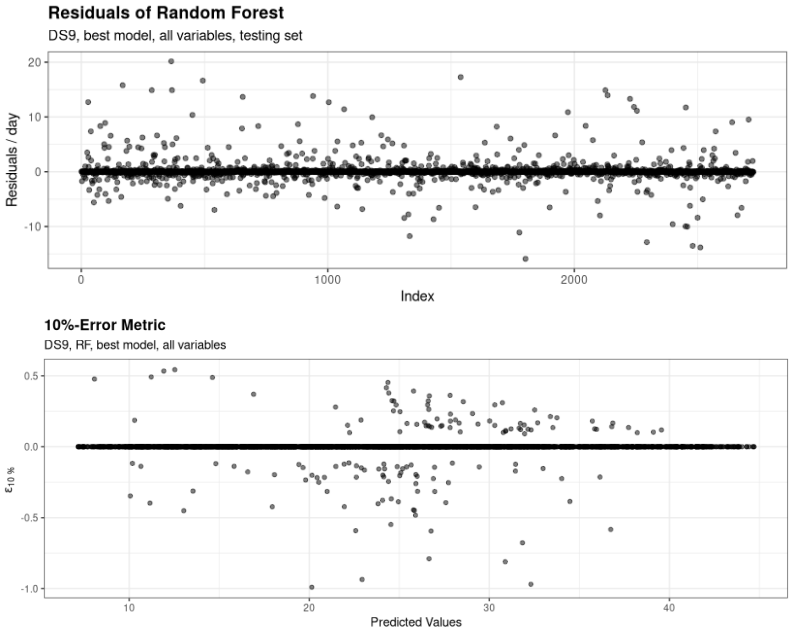

Figure B.13: Residuals (top panel) and 10 %-error metric for the RF model trained with all available variables in DS9.

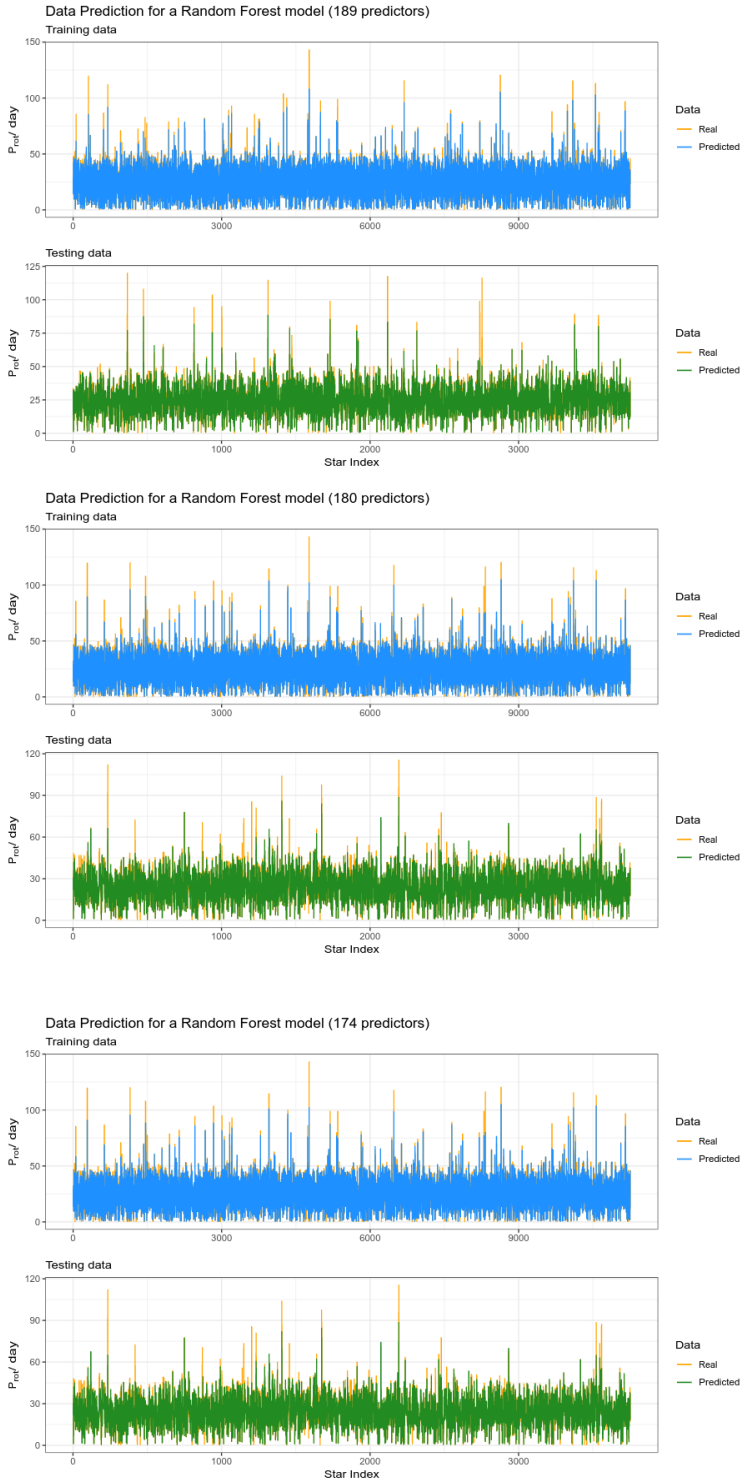

Figure B.14: Predictions on the training and testing sets for RF models with 180 (top), 171 (middle), and 165 (bottom) predictors. Orange lines correspond to the ground truth, while blue ones indicate predictions on the training set and green lines on the testing set.

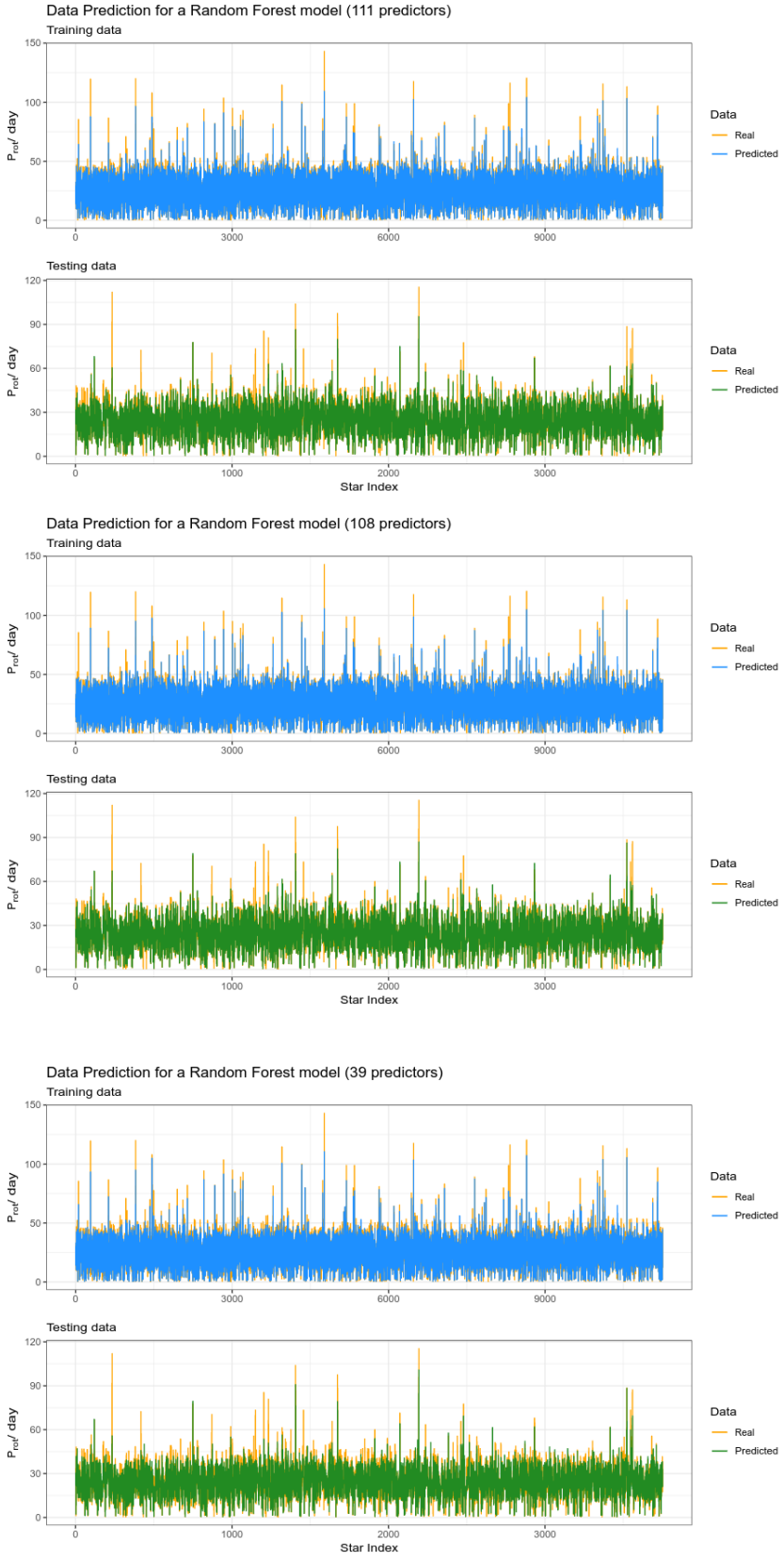

Figure B.15: Similar to fig. B.14, but for models with 108, 102, and 39 predictors.

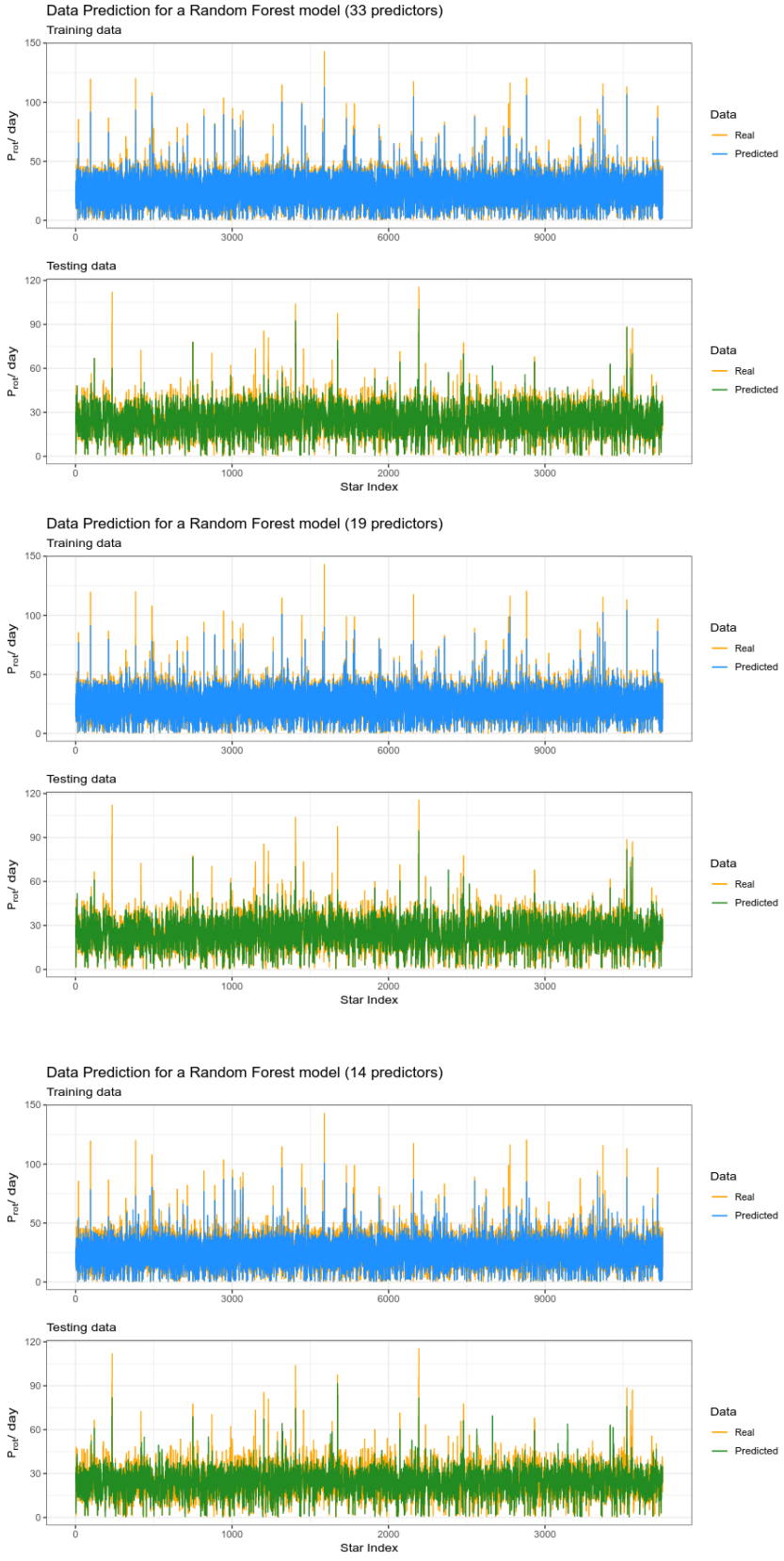

Figure B.16: Similar to figs. B.14 and B.16, but for models with 33, 19, and 14 predictors.

Index