University of Duisburg-Essen
Faculty of Business Administration and Economics
Chair of Econometrics

Statistical analization of professional road cyclists training data

Master Thesis

Submitted to the Faculty of
Economics
at the
University of Duisburg-Essen

from:

Nils Paffen

Reviewer:	Prof. Dr. Christoph Hanck
Second Reviewer:	Prof. Dr. Andreas Behr
Deadline:	04.05.2022

Name:	Nils Paffen
Matriculation Number:	3071594
E-Mail:	nils.paffen@stud.uni-due.de
Study Path:	M.Sc. VWL
Semester:	7^{th}
Graduation (est.):	Summer Term 2022

Contents

List of Figures

List of Tables

List of Abbreviations

Abstract

Professional road cyclist try to improve several attributes during their training session. One of those is the average power expressed in watt which can be interpreted as an performance indicator of the athlete. In recent findings, such as Karetnikov (2019), Kholkine et al. (2020), tree-based ensemble methods, combining several tree-based models to one model in an additive sense, showed the best results when used to predict the average power of road cycling athletes compared to different linear models. We will investigate if this also applies to a dataset scraped form the website Strava.com, which enables amateurs and professionals to upload their training efforts measured by a bicycle computer. The latter data came with some values which were either missing or likely erroneous. Three different strategies, combining variables, imputing variables or discarding observations, were used to create 4 different versions of the Strava dataset. We analyzed how those strategies helped to predict the average power. Besides the latter measurement, we explored the possibility of predicting the UCI weekly points which summarizes the tournament and other race results of professional road cycling athletes. To improve the results of the tree-based methods we tweaked the hyperparameter settings using an Bayesian Hyperparameter Optimization (Bayesian HPO) approach and compared the findings to those achieved by default settings. Furthermore, we will present those hyperparameter settings that accomplished the greatest prediction results. With the help of SHAP values we will analyze the best model to shed light on the insights of these tree-based ensemble models, such as variable importance and influence on the predictions.

The program code, which was used to generate the results of this thesis, can be downloaded under the following link : https://github.com/Npaffen/Master_Thesis/

1 Introduction

One of the oldest forms of competition in the modern professional sports industry is road cycling. Mignot (2015) claims that with the beginning of the late nineteenth-century the public interest in professional road cycling events such as the Bordeaux-Paris race in 1869 with a distance of 572 kilometers started to rise. The first Tour de France was organized in 1903 with a distance of 2,448 kilometers, which nowadays is one of the largest sporting events to take place annually. To prepare for these kind of tournaments many professional road cyclists share their training efforts with the public via Strava.com. For this thesis we collected training data of 185 professional road cyclists from 19 teams of the UCI World Teams list 2020. We added personal data, such as age or height, from procyclingstats.com and tournament points, from sprinter and climber events, to create a variable which reflects the $type$ of an athlete. We used this dataset to investigate if we can predict the average power of training sessions and if we can predict the weekly updated UCI weekly points. The latter points were awarded to cyclists for their results in tournaments or races. Furthermore we will use the IRMI algorithm to impute missing data or data which we argue are likely measuring errors. The latter strategy of handling missing values will be compared to a more conservative approach e.g., excluding such observations. A variety of tree-based ensemble methods, such as RandomForest or XGBoost, will be used to create a model from the dataset to predict the mentioned variables of interest. Since the latter models often contain several hyperparameter settings to specify the model training process, such as the number of trees for each model or a learning rate, we will use a Bayesian Hyperparameter optimization (HPO) algorithm to find the best hyperparameter settings for each model and method. A recent game-theory based approach called SHAP values for tree ensembles (TreeSHAP) Lundberg et al. (2018) is used to shed light on the tree-ensemble models which achieved the best prediction results. We are interested in answers for the following research questions :

- **Which tree ensemble method is the best to predict the average power and the UCI weekly points of the dataset?**
- **Can imputation strategies such as IRMI improve the prediction results?**
- **Which parameters are favorable/unfavorable for the prediction of the average power and the UCI weekly points?**
- **What are meaningful metrics to evaluate and compare models fitted with tree ensemble methods?**

- **Does Bayesian HPO improve the prediction ability of a tree ensemble method?**
- **If so, what are the optimal hyperparameter settings for the best model?**
- **Can we use TreeSHAP values to interpret the impact of the variables on the prediction results of the best model?**

This thesis will begin with an description of the scraping process of the sources like Strava.com which follows an overview of the dataset. A presentation of the imputation strategys we used for missing values in the Strava dataset The following literature section will describe research results from the the professional road cycling field. In section 4 we will explain how a selection of tree-ensemble methods use decision trees to create a model for the prediction task. In the same section we will dive into the different hyperparameter settings of the tree-based algorithms and how we can optimize those using a Bayesian HPO algorithm. To compare the model results, the metrics subsection will answer the question which metrics are preferred given the chosen methods to select the best model. A summary of SHAP values is followed by a description of four different models which we will use to predict the average power and the UCI weekly points.

2 Data

2.1 Scraping process

For the data analysis in the upcoming chapters, a dataset of training sessions from professional road cyclists, further mentioned as Strava dataset, was scraped from Strava.com. Besides the prediction of the average power we want to research if we can predict the UCI World ranking of the individual cyclists. Therefore we obtained the weekly UCI World Points for all available cyclists from https://firstcycling.com/ and merged these with each individual training observation of each athlete. The Strava dataset was collected by a scraper written in *C#*, which exploited a technical loop-hole of the website. The choice of coding a third-party application with *C#* instead of using implementations in R, is an not automatable log-in page[1]. The log-in procedure forces the user to press a button which is not accessible via a script or a simulated mouse click. To the best of the authors knowledge the only way to deal with this button is by pressing it when the site is fully loaded and graphical access

[1] Strava Log-In

of all features of the log-in page from Strava are possible. To bypass the log-in page we build a modified open-source version of the Google Chrome browser using the *C#* library *CefSharp*. Adjustments that were made are an integrated scraping-tool which used the script language CommandInterpreter to generate *.ci files which stores information such as the hyperlink to an athletes training activity and the Strava-ID of the same athlete.

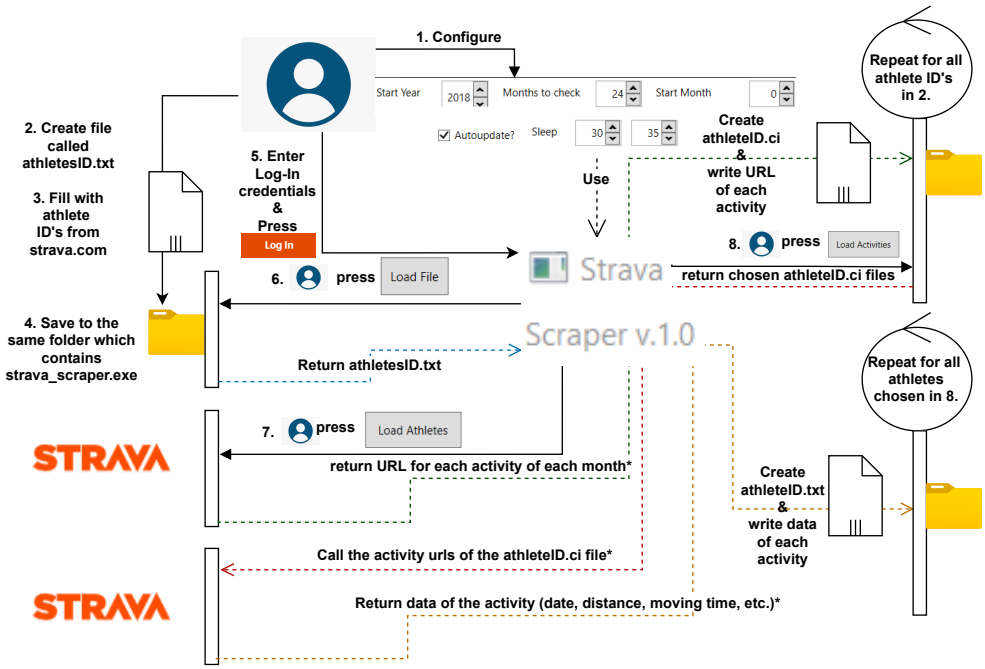


Figure 1: Diagram of the working process of the Strava scraper. All user interactions are marked by full line arrows, all automized interactions by the Strava scraper are highlighted by coloured dotted arrows. Colours were added just for visual purpose. "*" - signs at the end of a description indicate that this process interacts with Strava.com. Therefore a sleep time after each url call is used to prevent a timeout.

Figure 1 explains in detail how the scraping process with the Strava scraper works. The order of user interactions is structured by numbering. Some log-in credentials were hard-coded and could be accessed by pressing the buttons 'Copy Username' and 'Copy Password'. A picture of the application can be found in the appendix 19.

Scraping data from procyclingstats.com was much simpler due to its less restrictive accessibility. The website still misses an API but in this case a R-Script based on the *rvest* package collected some personal athlete data such as birthday or height. The same method was used to obtain the weekly UCI World points from firstcycling.com.

2.2 Dataset exploration

The Strava dataset contains training sessions from 185 professional road cyclists from the UCI World Teams list which featured 19 teams in 2020. The Strava dataset contains 61840 observations. Table 1 presents an overview of all variables of the Strava dataset. 18 variables were obtained or generated from training activities of the Strava website between 2018-01-01 and 2021-03-18. The variables *age*, *height*, *climber_points*, *sprinter_points* were obtained by data from procyclingstats.com. The two latter variables are the points an athlete achieved for their race results in climbing or sprinting competitions respectively. These variables were then used to create a new variable *type*. Each observation of the Strava dataset was matched with the aggregated point score from the first day of the year to the activity day of the year for each athlete gained from climbing and sprinting races respectively. Let d_i be defined as the date of an observation i of the Strava dataset of athlete k, D is the date of the observation for which the categorization $type$ was determined and $D \geq d$ is valid. Then, we can define the $type$ of each athlete k at each training activity i_D by the following rule :

$$type_{i_D,k} = \begin{cases} climber & \text{if } \sum_{i_d}^{i_D} climber\_points_{i_d,k} > \left\{\sum_{i_d}^{i_D} sprinter\_points_{i_d,k}\right\} * 1.25 \\ & \text{and } \left\{\sum_{i_d}^{i_D} climber\_points_{i_d,k} - \sum_{i_d}^{i_D} sprinter\_points_{i_d,k}\right\} \geq 30 \\ sprinter & \text{if } \sum_{i_d}^{i_D} sprinter\_points_{i_d,k} > \left\{\sum_{i_d}^{i_D} climber\_points_{i_d,k}\right\} * 1.25 \\ & \text{and } \left\{\sum_{i_d}^{i_D} sprinter\_points_{i_d,k} - \sum_{i_d}^{i_D} climber\_points_{i_d,k}\right\} \geq 30 \\ mixed & \text{else} \end{cases} \quad (1)$$

Therefore a cyclists of type $mixed$ is either a cyclists who achieved equally good results in both competitions until this point in time or participated in a few or none competitions before 2018. Since the attempt to classify a professional cyclists $type$ was never tried before, the ruling is based on some assumptions by the author. We expected that if the points of one of the competition types is larger than 1.25 times the other competition types, we expect that the athlete competes overall more in former competition than in the latter. We claim that this is enough to distinguish between a cyclists of type $mixed$ and either $climber$ or $sprinter$. To secure that the categorization of (young) athletes with a low number of competition results, is not heavily influenced by some singular outcomes, a minimal difference of 30 points between climber and sprinter competition results is required to be categorized as either $climber$ or $sprinter$. If an athlete fails to meet one of these criteria the athlete is attributed the $mixed$ type.

Table 1: Variable list of the Strava dataset

Variable name Description	Variable type
date : The date of an training observation.	*factor*
*avg_power** : Average power in watt generated during the ride.	*numeric*
avg_power_weig : Adjusted avg. power of the ride where a Strava algorithm corrects possible outliers in the data due to environmental impacts such as terrain, grade, wind and possible other factors not disclosed on the Strava website.	*numeric*
estAvgPower : Another estimate of the avg power of an activity generated by Strava. No further information about calculation were found.	*numeric*
*distance** : Distance of the training session in km.	*numeric*
*elevation** : Amount of meters of the route which contains elevation.	*numeric*
*avg_power_comb** : Generated variable by combining *avg_power*, *avg_power_weig* and *estAvgPower*.	*numeric*
*work_total** : The sum of watts generated during the ride, expressed in kilojoules(kJ).	*numeric*
*training_load** : Indicator for how long an athlete should rest after an activity. Power of a ride is compared to an individual functional threshold power. The latter is defined as the highest power output a road cyclist can preserve in a semisteady state for approximately 60 minutes.	*numeric*
*intensity** : An indicator to express the level of difficulty of a ride compared to the functional threshold power.	*numeric*
*avg_speed** : Average speed of the ride.	*numeric*
*max_speed** : Max speed of the ride.	*numeric*
*avg_cadence** : Average pedaling rate, average number of revolutions of the crank during the activity.	*numeric*
*max_cadence** : Max pedalling rate, maximal number of revolutions of the crank during the activity.	*numeric*
*avg_hearRate** : Average heart rate during the ride.	*numeric*
*max_heartRate** : Max heart rate during the ride.	*numeric*
*max_power** : Max power in watt generated during the ride.	*numeric*
*avg_calories** : Average calories burned during the ride.	*numeric*
*avg_temperature** : Average temperature during the ride.	*numeric*
*avg_elap_time_sec** : Average elapsed time in seconds during the ride.	*numeric*
*age** : The age of the athlete.	*numeric*
*type** : Categorical variable indicating if an cyclists is either of type 'climber','sprinter' or 'mixed'	*factor*
*height** : The height of the athlete.	*numeric*
*season** : Season of the year of the activity : spring, summer, autumn, winter.	*factor*
*UCI_points_weekly** : Indicates the weekly UCI points an athlete had on the in that week of the training activity.	*numeric*

Table 1 gives an overview of all variables of the Strava dataset. Those variables that have a * were used in the regression models, as explained in 4.10.

The measurement avg_power is the actual measurement of the average power provided by the bicycle computer and will be used as one variable to predict the average power of the Strava Dataset. avg_power_weig is the adjusted avg. power of the ride where an algorithm from Strava.com corrects possible outliers in the data due to environmental impacts such as terrain, grade, wind and other factors. The variable $estAvgPower$ is a guess of the average power measurement from Strava.com if there is no power data supplied by the bicycle computer. Karetnikov (2019) argued that a mean power threshold below 100 is

unreasonable and should be skipped. Therefore we excluded every observation where avg_power or $average\_power\_combined$ were lower than 100 watt due to possible negative influence on the prediction models. To maintain as many observations as possible of the Strava dataset, we decided to choose those where none of the three average power measurements showed a value below 100 watt. So that avg_power_comb was manufactured in the following sense :

$$avg\_power\_comb_j = \begin{cases} estAvgPower_j & \text{if} \quad avg\_power_j < 100 \\ & \text{and} \quad avg\_power\_weig_j < 100 \\ & \text{and} \quad estAvgPower_j \geq 100 \\ avg\_power\_weig_j & \text{if} \quad estAvgPower_j < 100 \\ & \text{and} \quad avg\_power_j < 100 \\ & \text{and} \quad avg\_power_j \geq 100 \\ avg\_power_j & \text{else} \end{cases} \tag{2}$$

avg_power_comb will be used as a second prediction variable for the average power measurement in a separated model from the avg_power variable. Models that contain the avg_power_comb variable consist of more observations compared to those that contain the original measurement avg_power.A detailed description of all models will be presented in section 4.10 We will see in the section 5 how good those estimates can help to predict the average power instead of omitting those observations of the avg_power variable that do not meet the 100 watt threshold.

2.3 IRMI

We found that 6880 observations in the Strava dataset contained missing values for either the $avg\_temperature$ and/or $avg\_calories$ variable(s). An attempt to deal with missing values is to impute them using variables which were observed or calculated directly from the bicycle computer. . To handle this problem we decided on two different strategies. First, the observations which contained NA values were dropped, second those values were instead imputed. The imputation was implemented using the IRMI algorithm mentioned first by Templ et al. (2011). The basis for the work of the previous mentioned authors is the IVEWARE algorithm from Raghunathan et al. (2001) which generates itera-

tive estimates for the missing values using a chain of regression models and picking values from the generated predictive distributions. The IRMI algorithm solves the inability of the IVEWARE algorithm to produce robust results for data including outliers, adds more flexibility by removing the restriction of at least one fully observed variable. In the latter process an user-specified amount of the most important variables for the imputation sequence were chosen. A short example of the algorithm for both variables of the Strava dataset then contain missing values is explained next.

Consider a dataset with n observations and m features $q_1, \ldots, q_m$. $\mathscr{D} = \{(\mathbf{x}_i, y_i)\} \left(|\mathscr{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}\right)$ In the first step a KNN-algorithm is used to create initial values for those missing values. Then, for each variable j of the dataset, those observations that contained missing values were marked. With $q \in \{1, \ldots, m\}$ we define

$$A_q := \{x \in \mathscr{D} | x^{(q)} = NA\} \tag{3}$$

So A_q describes all observations of the variables q that contain missing values. Then linear regression estimates $\hat{\beta}^j$ were calculated by all other variables including an intercept $X^j_{A,q}$ against the missing values of the target variable $y^m_{A,m}$. So that $y^m_{A,q} = \hat{\beta}^j X^j_{A,q} + \epsilon$, with ϵ being some error term. Then, we can replace (new) estimates for the missing values by $\hat{y}^m_{A,q} = \hat{\beta}^i X^i_{A,q}$. Afterwards the regression and replacing steps were repeated $min(2, M) \quad m \in M$ times until $\sum_a^A \left(\hat{\boldsymbol{y}}^m_{a,m} - \tilde{\boldsymbol{y}}^m_{a,m}\right)^2 < \delta, \quad \text{for all} \quad a \in A \quad \text{and} \quad m \in M$. Where δ is a small constant, $\hat{\boldsymbol{y}}^m_{a,m}$ is the a-th imputed value of the current iteration and $\tilde{\boldsymbol{y}}^m_{a,m}$ is the a-th imputed value of the last iteration. In section 5 we will explore if an improvement of the results can be achieved with imputed missing values in the dataset. The baseline will be the Strava dataset which excluded the observations that contained the missing values and one with imputed values generated by the IRMI algorithm for the models containing avg_power and avg_power_comb.

To impute variables with NA values we chose those 5 variables from the Strava dataset, that have the highest absolute correlation with the variables to be imputed. Therefore the regression models we constructed are presented in equation 4 and 5.

$$\begin{aligned} avg\_calories = {} & \underset{(0.911)}{mov\_time\_sec} + \underset{(0.899)}{distance} + \underset{(0.866)}{work\_total} \\ & + \underset{(0.764)}{elevation} + \underset{(0.526)}{max\_speed} + \epsilon \end{aligned} \tag{4}$$

$$avg\_temperature = \underset{}{max\_speed} + \underset{(-0.068)}{elevation} + \underset{(0.096)}{distance} + \underset{(0.066)}{distance} + \underset{(0.066)}{max\_heartRate} + \underset{(-0.055)}{\epsilon} \tag{5}$$

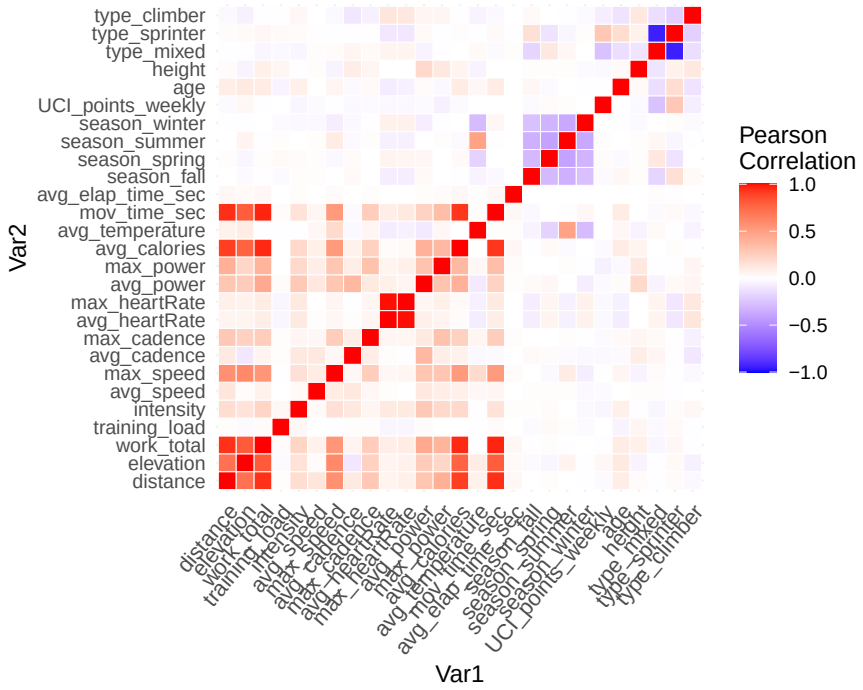


Figure 2: Heat map for the dataset with average power and dropped NA values.

Figure 2 shows the Pearson correlation coefficient in a heat map. Tiles which are more reddish implicate a high positive correlation, white tiles implicate that the two variables are uncorrelated while more bluish tiles mark a strong negative correlation. The variables we want to predict are avg_power and UCI_points_weekly. The variables with the highest correlation for the avg_power were $work\_total$(0.44), $avg\_calories$(0.4) and $avg\_cadence$(0.37). For UCI_points_weekly we observed a slightly negative correlation with $type\_mixed$ and a small positive correlation with $type\_sprinter$. Therefore we observed that those athletes who focus more on sprint races or tournaments with many sprint sections seemed to achieve higher UCI points, while athletes of $type$ mixed tended to achieve worse results. The overall low correlation with other variables indicated that a prediction of UCI_points_weekly with the Strava dataset might not achieve good results.

3 Literature

This section will provide an overview of findings of other authors using ensemble methods to predict average power measurements for professional road cyclists. Karetnikov (2019) used training data from a professional team and race data from procyclingstats.com to predict a mean maximal power (MMP) attribute for the upcoming race of the two top riders of the professional team. The author aggregated those values choosing several time frames, between 5 and 45 minutes, for the training and race data which were observed by bicycle computers. Another focus was on training and race data in mountainous areas defined as segments above 1500m. A total of 12 prediction techniques, such as LSTM, Decision Trees, Random Forests, XGBoost, CatBoost and 7 different regression models, were used to compare the RMAE as evaluation metrics. The XGBoost model achieved the highest prediction quality of 93% for the mountain stages and 88% for the flat stages. While an approach predicting starting positions in races was unsuccessful. A MC-study to identify if the XGBoost and Random Forest tree models depend on the chosen random seed showed that no influence on the prediction quality could be observed.

The research goal of Kholkine et al. (2020) was to predict race results of the 2018 and 2019 editions of the Tour of Flanders. The authors, approach was to consider the relative finishing time of several races in combination with an ten year exponential moving average and an aggregation of a variety of point achievements such as sprint race points. Team points of each cyclist were gathered from the previous year to account for possible team advantages. Kholkine et al. (2020) chose XGBoost as the predicting algorithm, since it showed high accuracy in past frameworks and can handle missing values. Imputation was not an option due to the variety of possibilities that an observation is missing. The model predicted 6 riders to be part of the top 10 and 2 cyclists out of top 3 in the 2018 edition correctly but the predicted rank was only correct for one cyclist. The predicted results for 2019 differed heavily from the actual results. No rank was predicted correctly but 4 riders could be properly identified to be part of the top 10. In the end, a comparison to fan prediction showed that the model could beat the fans, predicting expertise in 2018. A feature importance analysis based on the amount of splits each variable was used each time the dataset was divided showed that previous results of the *Omloop Het Nieuwsbald* and the *Tour of Flanders* were key features to the prediction model.

Passfield et al. (2016) reviewed how power meter output can be evaluated. Several approaches were considered such as smoothing data with a 30 second moving average, mean power output, the maximum mean power output ap-

proach, as used by Karetnikov (2019), and a critical power model. The latter is defined by a quite intuitive formula $(P - CP)\,t = W'$. The continuous power output is defined as P, CP is the critical power, t is time and W' is anaerobic capacity. A power output-duration curve can be constructed afterwards. The authors argue that a simple mean power output can be identical for different stressful training or race sessions and therefore does not reflect the variability of power output of each session precisely enough. Furthermore, Passfield et al. (2016) mention that instead of standard deviation, a detrended fluctuation analysis (DFA) looks more promising to examine power output variability due to its long-range correlations in time-series data.

A comparison between a machine learning model and a mathematical model prediction was made by Lemaître and Lemaitre (2018). The authors gradient boosting machine algorithm delivered better prediction results in terms of MAE than the model based on mechanics but stressed that the former had problems to correctly identify short power peaks.

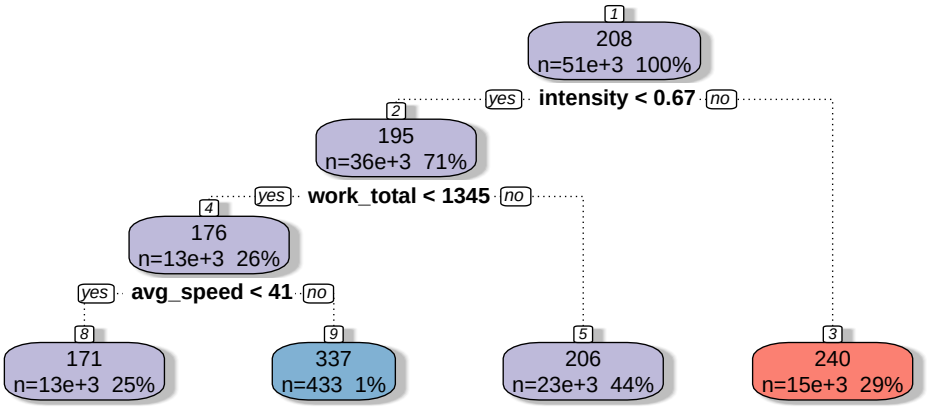
4 Methods

4.1 Regression Trees

Decision trees are the most basic tree-based models which are part of the non-parametric algorithms class. The latter means that we do not estimate a direct influence of a variable on the model. The algorithm divides a multi-dimensional feature space into unique sets, so called nodes. Generally, we distinguish between two decision tree tasks, classification and regression trees. Since our y, the variable of interest, avg_power, avg_power_weekly and UCI_weekly_points are all continuous variables, we will just focus on regression trees. Each split in a regression tree is ruled by the decision of a variable threshold. At the beginning, there is a root node, which contains the entire dataset and implies the first split into subsets. Each following node from a split can be either a decision node or a terminal node. The further term is used for nodes that will lead to a further split of the dataset. If no further split happens at one node, it is called a terminal node, leaf node, leaf. To decide which variable x and which threshold j of this variable x should be chosen to split the dataset, one uses the sum of squared residuals to evaluate the possible splitting decision for a regression trees at a node t.

$$SS_t = \sum_{i}^{n_1} (y_1 - \bar{y}_1)^2 + \sum_{i}^{n_2} (y_2 - \bar{y}_2)^2 \tag{6}$$

with $y_1 = (y_i | x_i >= j)$, $y_2 = (y_i | x_i >= j)$. We calculate the SS_t for each variable x and each threshold j of each variable x of the dataset. Then we chose the variable-threshold combination that generates the lowest SS_t as our splitting criterion.



Decision Tree Example for the Strava irmi dataset

Figure 3: Decision Tree Example for the Strava irmi dataset

Figure 3 shows an example of a regression tree with $y = avg\_power$ and X containing all variables with a star from 1 despite the avg_power_comb variable. In our example, the split at the root node divided the dataset into subsets, that had a value for the variable $intensity < 0.67$, indicated by *yes*, and those that do not, *no* in the aforementioned figure. The dataset is divided at each decision node, by evaluating all possible thresholds of all variables of the dataset or the subset of the dataset. This procedure is continued, until a further decision would fail to reduce the variance by a specified amount called *complexity parameter*. In figure 3 this parameter is set to 0.05. So the regression tree only performs a further split if the variance is reduced by at least 5% in the created node.

4.2 Random Forests

Estimating only one decision tree might give us a to narrow solution for the prediction of our target variable. Breiman (2001) showed that generating several uncorrelated decision trees , in terms of their prediction error, should give on average a better model to predict from than a model from a single tree. This is

due to the convergence of the error of the aggregated random forest model. So with an increased number of unique decision trees, the error of the random forest model should converge to the mean average error. To ensure that we aggregate a model of decision trees that have a low correlation between each tree, two methods were used. With bagging, a dataset of n observations is drawn with replacement from the dataset. Decision trees are very sensitive to (small) changes in the dataset, since decision rules are based on singular values of a variable. If the value of a decision rules is missing, e.g. due to bagging, a new tree structure is possible, since all subsets after this decision node were affected. The other method is feature subsampling. If we only choose some features of the dataset instead of all and vary those chosen features with each new decision tree, the possibility to end up with an decision tree that is correlated to some other tree from the dataset should be low, and thereby the correlation of the prediction error should be low. To predict the value of the target variable of a new observation, we collect the classification result of each tree in the random forest and choose the class which was predicted by the majority of trees.

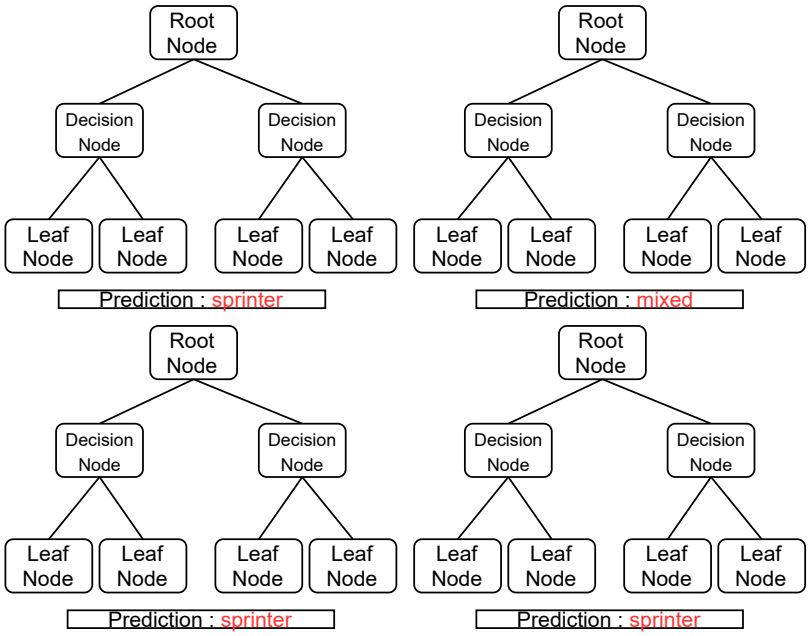


Figure 4: Diagram of a random forest prediction example. A new observation is shown to the model and each tree gives its prediction on the target variable, here $type$, of the Strava dataset. The figure is just for clarification of the concept and does not necessarily represent a possible outcome of a random forest model.

Figure 4 gives an example of a random forest prediction. The latter prediction

of the target variable $type$ would be $sprinter$, Since 3 out of 4 trees would predict that the new observation would be of type $sprinter$.

4.3 Gradient Boosting

Mason et al. (1999) proposed an boosting algorithm based on gradient descent. Gradient Boosting Models generate multiple trees and add them together to generate a single ensemble model. The Gradient Boosting machine (GBM) learns with each iteration which the splits worked in terms of a regularized learning objective. Consider some input data $\{(x_i, y_i)\}_{i=1}^{n}$ and a differential loss function $L(y, F(x))$. $F(x)$ is a model to predict $\hat{y}$. A loss function is an evaluation metric of a model $F(x)$ and our target variable y. As an example we will use

$$L(y, F(x)) = \frac{1}{2}(y - \hat{y})^2 \tag{7}$$

as our loss function. Where the $F(x)$ is a model to predict $\hat{y}$. The next section 4.8 will discuss different loss functions. One starts with an initial guess $F_0(x)$ as the prediction of the target variable y, defined as γ. Formally

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, \gamma\right) \tag{8}$$

where argmin means that we need to find a γ that minimizes the sum of the equation. Considering the derivative of our loss function, we solve for γ so that

$$\frac{\partial L\left(y_i, \gamma\right)}{\partial \gamma} = \frac{\partial \sum_{j=1}^{n} \frac{1}{2}(y_j - \gamma)^2}{\partial \gamma} = \sum_{j=1}^{n} -y_j + \gamma = 0 \tag{9}$$

$$\gamma = \frac{\sum_{j=1}^{n} y_j}{n} = \overline{y} \tag{10}$$

the optimal γ for $F_y(x)$ is equal to the mean of y. In other words, the initial prediction $F_0(x)$ is a leaf, a constant value, that predicts the target variable y. Afterwards, the prediction error r is obtained as before by subtracting the real value y from the prediction of the target variable γ. So that, for $m = 1 \quad to \quad M$, where M is the maximum number of trees GBM should build, we compute the prediction errors as

$$r_{im} = -\left[\frac{\partial L\left(y_i, F\left(x_i\right)\right)}{\partial F\left(x_i\right)}\right]_{F(x)=F_{m-1}(x)} \text{ for } i = 1, \ldots, n \tag{11}$$

. If $m = 1$ equation 11 calculates prediction errors for each observation $i = 1, \ldots, n$ of the dataset for the case of $F(x) = F_0(x)$. Considering the minus in

front of the derivative of the latter case, one ends up with $(y - F(x))$, which is equal to the function of a residual. If one uses another loss function, one ends up with a function similar to the residual function but not quite. Therefore, one defines r_{im} as pseudo residuals. The gradient in equation 11 is the reason for the name of GBM. Now the GBM fits a regression tree, but this time instead of predicting the target variable y, we predict the pseudo residuals r_{im}. As before in the Random Forest algorithm, the GBM does not use all possible information of the dataset to build a tree but is restricted by a maximum number of leaf nodes for each tree L_m. So that $l = 1, \ldots, L_m$ The output values γ_{lm} of each leaf node R_{lm} are defined as

$$\gamma_{lm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{lm}} L\left(y_i, F_{m-1}(x_i) + \gamma\right) \quad l = 1, \ldots, L_m \tag{12}$$

where $L\left(y_i, F_{m-1}(x_i) + \gamma\right)$ is equation 8 expanded by $F_{m-1}(x_i)$, which defines the previous prediction of the the target variable y. $\sum_{x_i \in R_{lm}}$ defines that only those observations of the dataset were used for the calculation of γ_{lm} that match with the pseudo residuals in leaf l. In the end, due to our chosen loss function, the output values γ_{lm} were always the average of the pseudo residuals in leaf l. In a last step we will update our model with our new predictions so that

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{m} \gamma_{jm} I\left(x \in R_{jm}\right) \tag{13}$$

where ν is a learning rate which shrinks the influence of each tree m, so that overfitting is reduced, future trees $m+1$ can still improve the model and $\sum_{j=1}^{m} \gamma_{jm} I\left(x \in R_{jm}\right)$ adds up the output value γ_{lm} for all leaves, $R_{j,m}$ that refer to the pseudo residuals of tree m and observation j of the training dataset. GBM will stop building trees and update the pseudo residuals if either the number of trees m reached the maximum number of trees M or the shrinkage value of the pseudo residuals $R_{j,m}$ fall below a specified threshold.

Again we consider the variable avg_power_comb as our target variable. So we calculate the initial prediction value from our target variable, obtain the pseudo residuals as described before and use this pseudo residuals as 4 terminal nodes. Let the 3 splitting rules be $intensity > 0.69$, $avg\_speed > 35.9$ and $type = mixed$. Then calculate the output value at each leaf node and add the weighted results of this process to each observation of the target variable of each pseudo residual respectively.

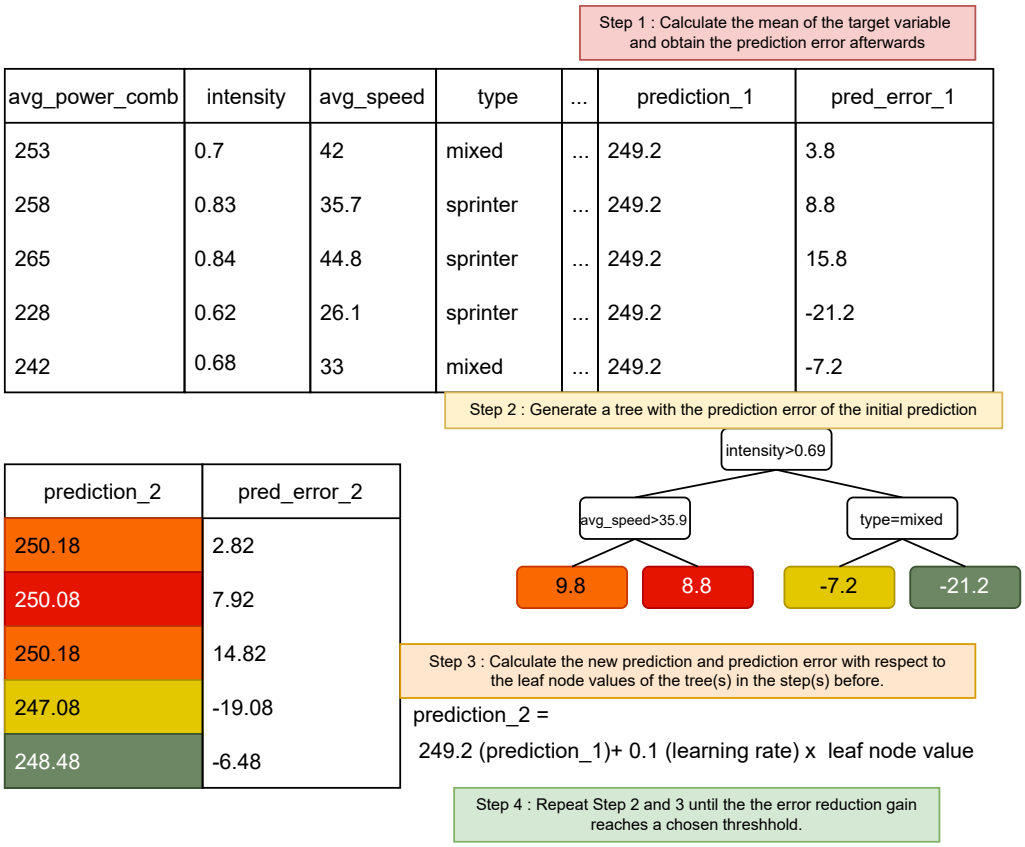


Figure 5: The diagram shows exemplary how GBM calculates the predictions.

Figure 5 shows a visualization of this process. We decided to not include the GBM method in the results section since XGBoost (section 4.4) and Lightgbm (section 4.5) are direct successors of this technique and are likely to outperform the GBM method. We still included this method to give a good introduction into the following gradient-based tree-ensemble methods.

4.4 XGBoost

The extreme gradient boosting (XGBoost) algorithm was first mentioned by Chen and Guestrin (2016).

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathscr{F} \tag{14}$$

$$\text{with } \mathscr{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\} \left(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T\right)$$

Equation 14 defines the predicted value $\hat{y}_i$ as the sum of values from K repetitions of $f(x_i)$. This means that the prediction of the target value y for an example i is given by the sum of predictions from K trees. 4.4 from Chen and

Guestrin (2016) shows the space of the regression trees. Let T be the number of terminal nodes in a tree. Then, q can be defined as the decision rules which creates the structure of a tree with a root node in the beginning and some terminal nodes in the end. w can be defined as the continuous score on the i-th terminal node. So the (final) prediction of $\hat{y}$ will be calculated by the sum of the corresponding terminal nodes which are given by w. Chen and Guestrin (2016) expand the loss function of GBM by some regulization paramter Ω to create a regularized task.

$$\begin{aligned} \mathscr{L}(\phi) &= \sum_i l\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \\ \text{where } \Omega(f) &= \gamma T + \tfrac{1}{2}\lambda \| w \|^2 \end{aligned} \tag{15}$$

In equation 15m the loss function is defined by $l\left(\hat{y}_i, y_i\right)$ and calculates the difference of the predicted $\hat{y}$ and the actual value of y. Ω controls the complexity of the model by adding a penalty parameter γ to alter the number of terminal nodes of a tree, and $\frac{1}{2}\lambda \| w \|^2$ is used to level the weights as a protection against overfitting.

To optimize 15, we need to train the model in an additive sense. This means, that in each instance i of the iteration t we predict $\hat{y}_i^{(t)}$ and will add f_t to minimize the objective. Using an second-order Taylor approximation, we can optimize the objective so that

$$\mathscr{L}^{(t)} \simeq \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}^{(t-1)}\right) + g_i f_t\left(\mathbf{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right) \right] + \Omega\left(f_t\right) \tag{16}$$

with $g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$. At step t, we can remove $l\left(y_i, \hat{y}^{(t-1)}\right)$ which is independent from $f(t)$.

$$\tilde{\mathscr{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t\left(\mathbf{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right) \right] + \Omega\left(f_t\right) \tag{17}$$

Let $I_j = \{i | q(x_i) = j\}$ be the instance set of terminal node j. An instance set can be explained as a subset of the dataset with an instance index i and their gradient statistics g_j, h_j for each observation i respectively. Expanding Ω in eq.

16, we end up with

$$\begin{aligned}\tilde{\mathscr{L}}^{(t)} &= \sum_{i=1}^{n}\left[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T\end{aligned} \tag{18}$$

At the $t-th$ iteration, we need to sum the weights w^2 of each terminal node. Doing so, we can replace $f_t(x_i)$ as the function to optimize, with w_j a parameter to optimize. For each terminal node j, we calculate G_j and H_j as the sum of each gradient statistic g_i and h_i for each observation i of the instance set I_j.

Let the structure of $q(x)$, be fixed, and one can derive the optimal weight w_j^* of terminal node j, as

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \tag{19}$$

.

w_j^* is then used to simplify the regularized learning objective $\tilde{\mathscr{L}}^{(t)}$ by expanding the object to a function depending on the tree structure q resulting in

$$\tilde{\mathscr{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{20}$$

.

Equation 20 can be interpreted as a scoring function and thereby as an evaluation metric of the tree structure q. XGBoost uses this function to prune the tree structure q in the sense that XGBoost chooses the structure q with the lowest score of $\tilde{\mathscr{L}}^{(t)}(q)$

Evaluate each possible split at a decision node, we would often end up with an amount of tree structures q that is impossible to evaluate. XGBoost therefore uses a greedy algorithm, so that a tree always starts with just the root node and iteratively adds more decision nodes. Let I_L and I_R be the instance sets of left and right nodes after a possible splitting decision, with $I = I_L \cup I_R$. The instance set of a splitting decision can be expressed as the instance set of the left and right node after a possible split. The loss reduction of a possible split candidate can be defined as

$$\mathscr{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{21}$$

.

Further methods used to reduce overfitting are shrinkage and column subsampling. Shrinkage scales new weights by a factor η succeeding each step of tree boosting. Thereby the influence of each individual tree is reduced and future trees have a higher possibility to enhance the model, so that one can consider η as a learning rate. Feature subsampling is used to reduce the complexity of each tree structure q. Chen and Guestrin (2016) propose a weighted quantile sketch algorithm to decrease the computational time needed to find the optimal splitting candidate at each node of the structure q for each iteration t. The discussion of this technique was omitted as it would go beyond the scope of this thesis.

Figure 6 shows a visualization of the process The implementation of the XGBoost algorithm sets the initial loss $\mathscr{L}^{(0)} = l(y_i, \hat{y}_i) = 0.5$. Again we consider the variable avg_power_comb as our target variable. So we calculate the initial prediction value from out target variable,otain the pseudo residuals as described before, and use these pseudo residuals as the dataset for the tree structure q. Let the 3 splitting rules be $intensity > 0.69$, $avg\_speed > 35.9$ and $type = mixed$. Then calculate the output value at each leaf node and add the weighted results of this process to each observation of the target variable of each pseudo residual respectively.

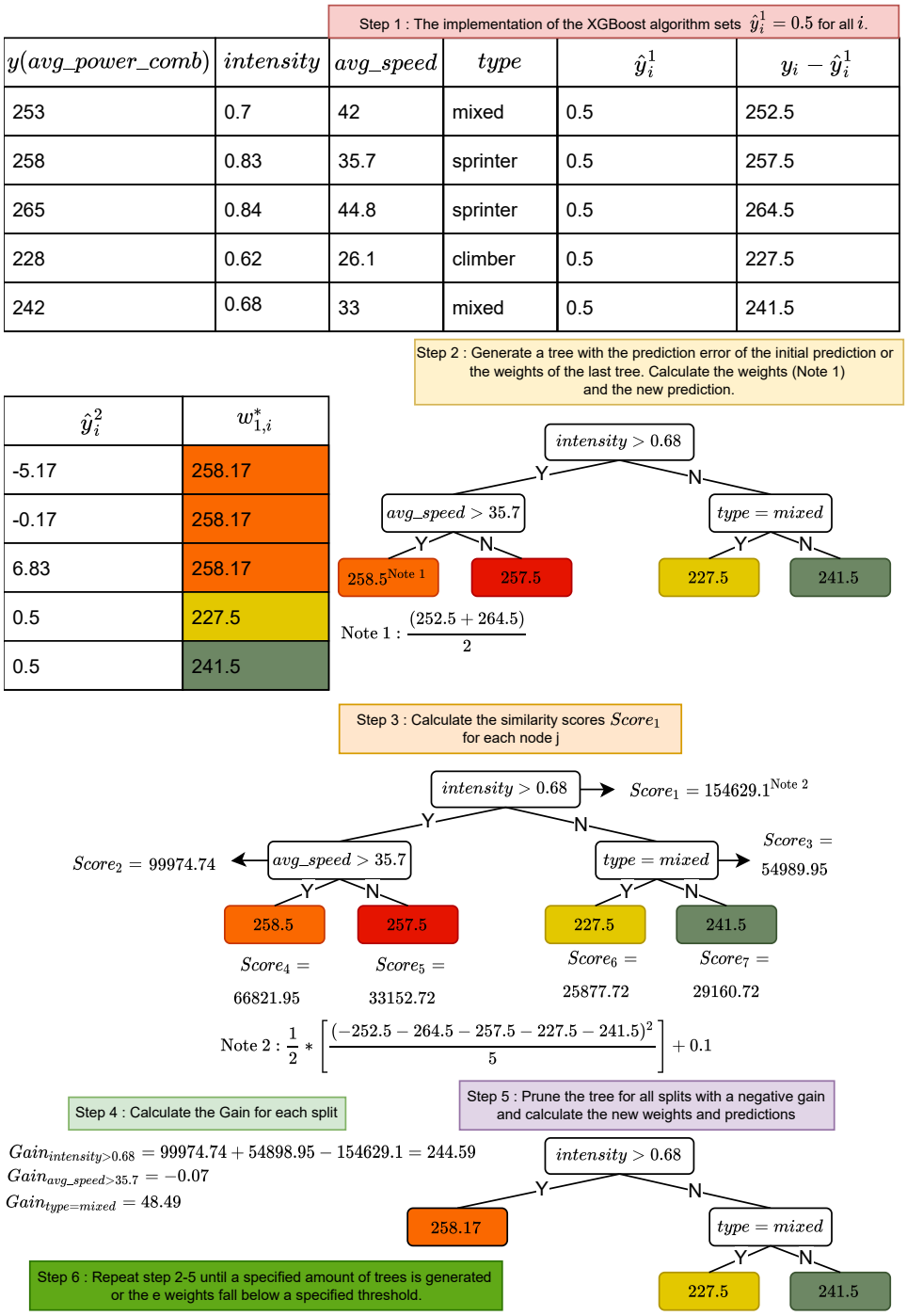


Figure 6: The diagram shows exemplary how XGBoost initiates the algorithm, builds trees, calculates the weights and updates the model.

4.5 LightGBM

Ke et al. (2017) proposed a gradient boosting algorithm called LightGBM (LGBM), which supports a way to differ between instances with small and large absolute gradient statistics. They call this method *Gradient-based One-*

Side Sampling (GOSS) and argue that those instances with a small gradient also show a low training error and should therefore receive less attention. GOSS sorts the instances with respect to their absolute value of their gradients in descending order and chooses the top $l \times 100\%$ instances. From the other $1-t$ share of instances, GOSS randomly samples $s \times 100\%$ instances. The latter samples are multiplied with a small weight $\frac{1-l}{s}$ when the loss function is evaluated.

The following explains the splitting decision for trees in LGBM. $\tilde{v}_j(p)$ is the estimated variance gain when we split a feature k at point p. As explained before, LGBM differs between instances with large and small absolute gradient statistics. So that $C_l = \{x_i \in C : x_{ik} \leq p\}$, $C_r = \{x_i \in C : x_{ik} > p\}$. The feature values smaller than or equal to the threshold p of those instances with large absolute gradient statistics would be split to the left child node and those exceeding the threshold p would be split to the right child node. The same definition holds for D_l and D_r, with the difference that D represents randomly sampled instances from those with already low absolute gradient statistics. Formally, LGBM estimates $\tilde{v}_k(p_k^*)$, because we train with a dataset of instances that is smaller than the dataset of all possible instances, such that

$$\tilde{v}_j(p) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in C_l} g_i + \frac{1-l}{s}\sum_{x_i \in D_l} g_i\right)^2}{n_l^k(d)} + \frac{\left(\sum_{x_i \in C_r} g_i + \frac{1-l}{s}\sum_{x_i \in D_r} g_i\right)^2}{n_r^k(pd)}\right). \tag{22}$$

Due to the lower amount of instances, LGBM reduces the computation time to find the best split candidate at each decison node. The other important difference of LGBM compared to XGBoost is exclusive feature bundling. Consider a sparse dataset with many dummy features, such that their distribution is binary. Ke et al. (2017) argued that those many features in a dataset with a sparse feature space are mutually exclusive, e.g. features that never share nonzero values for all observations. LGBM stacks those features into one feature *bundle* and thereby reduces the dataset dimension from $\#data \times \#features$ to $\#data \times \#bundle$ with $\#bundle << \#features$. This reduces the time to find splits since less features have to be considered.

4.6 Catboost

Prokhorenkova et al. (2019) proposed an enhanced gradient boosting algorithm called CatBoost. The latter is an abbreviation for categorical boosting. They

claim to be the first, who found a solution to the problem of target leakage in gradient boosting. The latter implies that due to the usage of the same training observations, for the calculation of the gradient statistics and models used to minimize those gradients, a prediction shift is produced. They also found a similar problem with the encoding of categorical variables which will be explained at the end of this section.

Considering a dataset $\mathscr{D}$ as described before for XGBoost, CatBoost wants to minimize the expected loss $\mathscr{L}(F) := \mathbb{E}L(y, F(\mathbf{x}))$ through an iteratively updated object F. The latter object will be approximated by F^t with $t = 0, 1, \ldots$ through the gradient boosting procedure as explained in GBM and XGBoost. Given that h^t is a function for a binary decision tree, as we minimize the expected loss by

$$h^t = \underset{h \in H}{\arg\min}\, \mathscr{L}\left(F^{t-1} + h\right) = \underset{h \in H}{\arg\min} \mathbb{E}L\left(y, F^{t-1}(\mathbf{x}) + h(\mathbf{x})\right). \tag{23}$$

Catboost takes the (negative) gradient step instead of the second-order Taylor approximation to solve the minimization problem, so that h^t approximates $-g^t(\mathbf{x}, y)$, with $g^t(\mathbf{x}, y) := \frac{\partial L(y,s)}{\partial s}\Big|_{s=F^{t-1}(\mathbf{x})}$. Prokhorenkova et al. (2019) then use a least-squares approximation so that:

$$h^t = \underset{h \in H}{\arg\min} \mathbb{E}\left(-g^t(\mathbf{x}, y) - h(\mathbf{x})\right)^2 \tag{24}$$

Equation 24 shows the ideal way to approximate h^t, but since we cannot know the expectation, an approximation often uses the same dataset $\mathscr{D}$ so that :

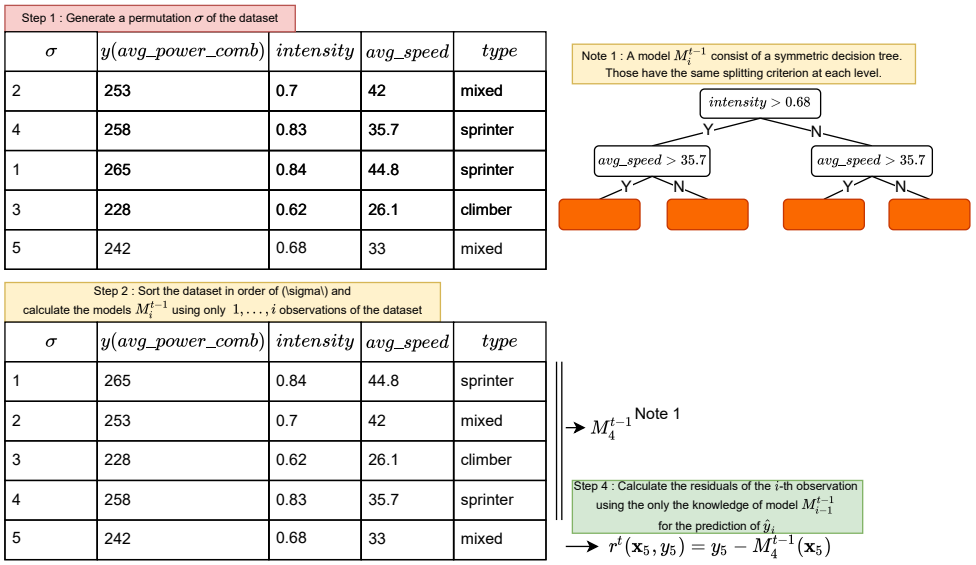
$$h^t = \underset{h \in H}{\arg\min} \frac{1}{n} \sum_{k=1}^{n} \left(-g^t\left(\mathbf{x}_k, y_k\right) - h\left(\mathbf{x}_k\right)\right)^2 \tag{25}$$

The authors now argue that the aforementioned prediction shift is due to a shift in $g^t\left(\mathbf{x}_k, y_k\right) \mid \mathbf{x}_k$ from $g^t(\mathbf{x}, y) \mid \mathbf{x}$, since F^t was trained using y_k. This implies that h^t is a biased estimate of $\hat{y}^t$ which leads to a bias in F^T, since the target values y_k of the same observations were used for all curent models F^{t-1}. As in equation 13 of GBM, CatBoost updates its prediction model in an iterative sense. The authors argue that this bias comes from a "shift of the conditional distribution $F(\mathbf{x}_k) \mid \mathbf{x}_k$ for a training example $\mathbf{x}_k$ from the distribution of $F(\mathbf{x}) \mid \mathbf{x}$ for a test example $\mathbf{x}$." Due to the latter, the learned model suffers from a (prediction) shift, so $F(\hat{y}_k | y_k)$ is shifted. CatBoost solves this by a technique

called Ordered Boosting.

Let I be the amount of trees a model is learned with. Unshifted residuals $r^{I-1}(\mathbf{x_k}, y_k)$ can only obtained if we exclude observation $\mathbf{x}_k$ in the training process of $FI-1$. The authors argue that the training process would become impossible because, since all residuals need to be unshifted, we cannot use any observations for the training process of $FI-1$. Prokhorenkova et al. (2019) solve this by a set of models which doe not include any information of the predicted observation. Therefore, one uses a permutation σ of $\mathscr{D}$ and learns n unique models $M_1,\ldots,M_n$. Each model M_i uses only observations in the permutation σ between the first and and the i-th observation. The residual of the j-th observation in the permutation σ uses the model M_{j-1}. CatBoost combines the latter method with gradient boosting with symmetrical decision trees to obtain unbiased residuals $R(\mathbf{x}_k, y_k)$ and thereby an unbiased combined model F.

Figure 7 shows how the ordered boosting algorithm works to create the combined model F. Since the calculation of the residuals and the iterative manner of gradient boosting techniques were explained in detail for GBM and XGBoost, figure 7 focuses more on the unique differences to other gradient boosting techniques.



σ	$y(avg\_power\_comb)$	$intensity$	avg_speed	$type$
2	253	0.7	42	mixed
4	258	0.83	35.7	sprinter
1	265	0.84	44.8	sprinter
3	228	0.62	26.1	climber
5	242	0.68	33	mixed

σ	$y(avg\_power\_comb)$	$intensity$	avg_speed	$type$
1	265	0.84	44.8	sprinter
2	253	0.7	42	mixed
3	228	0.62	26.1	climber
4	258	0.83	35.7	sprinter
5	242	0.68	33	mixed

Figure 7: The diagram shows how the orderd boosting algorithm in CatBoost works.

Categorical features have often been encoded by one-hot-encoding. This means splitting the (categorical) feature specifications by adding new binary variables. This can lead to a sparse dataset due to the large expansion of the feature space, if the categorical features have a high cardinalty, e.g. a userID

feature. Micci-Barreca (2001) proposed to tackle this problem using target statistics which estimates the expected target value for each feature specification, thereby grouping categories and using the target statistic as a new numerical variable instead. Prokhorenkova et al. (2019) claim that the prediction shift mentioned before applies in the same way when computing target statistics for a categorical feature. To avoid this, they propose a sub-sampling so that $\mathscr{D}_k \subset \mathscr{D}_{\{\mathbf{x}_k\}}$ is used to calculate the target statistics for x_k, therefore excluding x_k from the process. Let p be some prior to smooth the estimate $\mathbb{E}(y|x^i = x_k^i)$ for categories of a feature with a low proportion compared to the other categories of that feature, then :

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathscr{D}_k} \mathbb{1}_{\left\{x_j^i = x_k^i\right\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathscr{D}_k} \mathbb{1}_{\left\{x_j^i = x_k^i\right\}} + a} \tag{26}$$

Prokhorenkova et al. (2019) then introduce a technique called ordered targeting statistic. Comparable to the online learning algorithm, we feed the model sequentially, using a random permutation σ of the training dataset. Thereby, the model observes in each iteration another new training observation of the permuted training dataset, but all other observations from past iterations will be used to compute the target statistic as well. More formally, for the training process, equation 26 will use $\mathscr{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$, and for testing, one uses the whole dataset $\mathscr{D}_k = \mathscr{D}$.

Table 2
Orderd Target Statistics Example

target	target	encoded target prior p = 0.05, a = 1	history	explanation
0	mixed	$\frac{0+0.05}{0+1} = 0.05$	1	$\frac{\text{No } \textit{mixed} \text{ with target = 1 in history + a} \times \text{prior}}{\text{No } \textit{mixed} \text{ in history + a}}$
1	sprinter	$\frac{0+0.05}{0+1} = 0.05$	2	$\frac{\text{No } \textit{sprinter} \text{ with target = 1 in history + a} \times \text{prior}}{\text{No } \textit{sprinter} \text{ in history + a}}$
1	mixed	$\frac{0+0.05}{1+1} = 0.025$	3	$\frac{\text{No } \textit{mixed} \text{ with target = 1 in history + a} \times \text{prior}}{\text{One} \textit{mixed} \text{ in history + a}}$
0	climber	$\frac{0+0.05}{0+1} = 0.05$	4	$\frac{\text{No } \textit{climber} \text{ with target = 1 in history + a} \times \text{prior}}{\text{No } \textit{climber} \text{ in history + a1}}$
1	mixed	$\frac{1+0.05}{2+1} = 0.35$	5	$\frac{\text{One } \textit{mixed} \text{ with target = 1 in history + a} \times \text{prior}}{\text{Two } \textit{mixed} \text{ in history + a}}$

Table 2 shows a less formal explanation of ordered target statistics. Consider each row of *target* and *type* as single observations of the exemplary dataset. The derivation of the value in column *encoded target* is explained in the column *explanation*. The column *history* describes the 'observed history' of the ordered target statistics calculation. Such that, for the value of row with $history = 3$, we have $target = 1$ and $type = mixed$. No 'observed history' exists with $target = 1$, and $type = mixed$, so $\sum_{\mathbf{x}_j \in \mathscr{D}_k} \mathbb{1}_{\left\{x_j^i = x_k^i\right\}} \cdot y_j + ap$ in equation 26 is still 0, but $\sum_{\mathbf{x}_j \in \mathscr{D}_k} \mathbb{1}_{\left\{x_j^i = x_k^i\right\}}$ is now 1 since the sum of the denominator is no conditioned

on the target value y_j. The latter is important, since Prokhorenkova et al. (2019) argue that a desired property of the target statistics feature calculation and the learning process of the model is the effective usage of all training data. Still, the oblivious counter of the fraction satisfies the requirement to counter the prediction shift. The authors called this bias target leakage, since recognizing the actual target value of the observation for which the *encoded target* calculation is performed, leaks information about the distribution of the target to the encoding process. Since this learned distribution might differ from the distribution of the target value in the test dataset, a biased prediction due to overfitting is possible.

4.7 Hyperparameter Optimization

To optimize the tree ensemble models we can optimize their hyperparameters. This subsection will introduce four hyperparameter optimization (HPO) methods and discuss the choice for this work. Furthermore, we will discuss which hyperparameters are possible and useful for each tree ensemble technique. Tree ensemble algorithm such as the gradient boosting tree implementations of XGBoost, Lightbm and Catboost have several hyperparameter which handle :

- **learning rate** - the contribution of each (tree) model, often a factor betwen 0 and 1
- **mtry** - defines how many variables will be used for each (tree) model
- **sample size** - defines the number or share of observations of the training dataset to be used in each (tree) model
- **tree depth** - the maximum amount of levels each tree can generate
- **trees** - the number of trees to build
- **minimal node size** - a minimum number of observations in a node needed to accept a new split at a decision node of a tree
- **loss reduction** - minimum loss reduction needed to accept a split at a decision node of a tree
- **iteration stop** - the number of trees without improvement before stopping the algorithm

The aim of HPO is to find a set of hyperparameter values for a (machine learning) method which minimizes the chosen loss function to evaluate the quality of the model on the validation dataset. Let the chosen loss function be some function $f(x)$ and x^* the set of optimal hyperparameter values, then HPO can be stated as :

$$x^* = \arg\min_{x \in \mathscr{X}} f(x) \tag{27}$$

4.7.1 Manual search

The most simple solution for HPO is to choose a parameter value for each hyperparameter and compare the result of the loss function to some other parameter values. This is repeated until a reasonable low loss function value is found.

4.7.2 Grid search

Grid search uses a user specified vector of possible values for each hyperparameter to be tuned. Then all possible combinations of hyperparameter values are evaluated and the combination with the lowest metric value is chosen. This leads to a large amount of possible combinations to evaluate and therefore to a possible long computation time while at the same time only finding a solution for values specified by the user.

4.7.3 Random search

This HPO method do not uses any kind user specification for the hyperparameter values. Random search randomly samples hyperparameter values from the range of each hyperparameter sets, evalutes them and continues until a specified amount of iterations is reached. The hyperparameter set which accomplished the lowest metric value is retunred.

4.7.4 Bayesian HPO

Bergstra and Bengio (2012) argue that the latter two methods show slightly better performance in terms of loss reduction per computational time than manual search. Still, both suffer from the problem that they cannot evaluate the chosen set of hyperparameter values after each iteration. This problem is solved by Bayesian HPO which uses an acquisition function as a prior for the hyperparameter set to evaluate next. Consider an objective function such as equation

15 from XGBoost and a dataset with n observations and d features, which are the hyperparameters of our objective function, so that $x_i = (x_{i1}, \ldots, x_{id})^T$ and $y_i = y_i(x)$ Either one initialize the optimization process with t hyperparameter sets with $t \in n$ or the algorithm randomly samples t of such sets w.r.t. the range of each hyperparameter. $D_0 = x_1, \ldots, x_n$ stores the hyperparameter sets for n trials in a $n \times d$ matrix X. Normalization of x_i achieves $x_i \in [0,1]^d$. Afterwards, each set i is used with the objective function to obtain a performance metric as the i-th observation of the dependent variable y_i. Those outputs are stored in the $n \times 1$ vector $Y = y(X) = (y_1, \ldots, y_n)^T$. Instead of continuing to evaluate the objective function for further hyperparameter sets, Bayesian HPO uses a surrogate model to estimate the function to be optimized. The model used in this implementation is the Gaussian Process (GP) model:

$$y(x_i) = \mu + z(x_i); \quad i = 1, \ldots, n \tag{28}$$

where μ is the overall mean and $z(x_i)$ defines the GP with $E[z(x_i)] = 0$, $Var[z(x_i)] = \sigma^2$ and $Cov(z(x_i), z(x_j)) = \sigma^1 R_{i,j}$. Let $y(X) \sim N_n\left(\mathbf{1_n}\mu, \Sigma\right)$, where $N_n\left(\mathbf{1_n}\mu, \Sigma\right)$ is a multivariate normal distribution with $\Sigma = \sigma^2 R$ defined through a correlation matrix R with elements R_{ij} and $\mathbf{1_n}$, a vector of length n with all ones s $n \times 1$. The algorithm uses the Gaussian correlation function from Ranjan et al. (2011):

$$R_{ij} = \prod_{k=1}^{d} \exp\left\{-\theta_k \left|x_{ik} - x_{jk}\right|^2\right\}, \quad \text{for all } i, j \tag{29}$$

with $\theta = (\theta_1, \ldots, \theta_d \in [0, \infty)^d$ defines a vector for the values of a hyperparameter set. Less formal, the GP estimates a function to model the hyperparameter sets and their estimated metric value. With Σ defines a confidence interval for this estimated function which covers a larger area where the function estimate is only given by very few observations and covers a smaller area where more observations were sampled so far. Thus we can say that a larger confidence interval implies a higher uncertainty of the actual function values in this area.

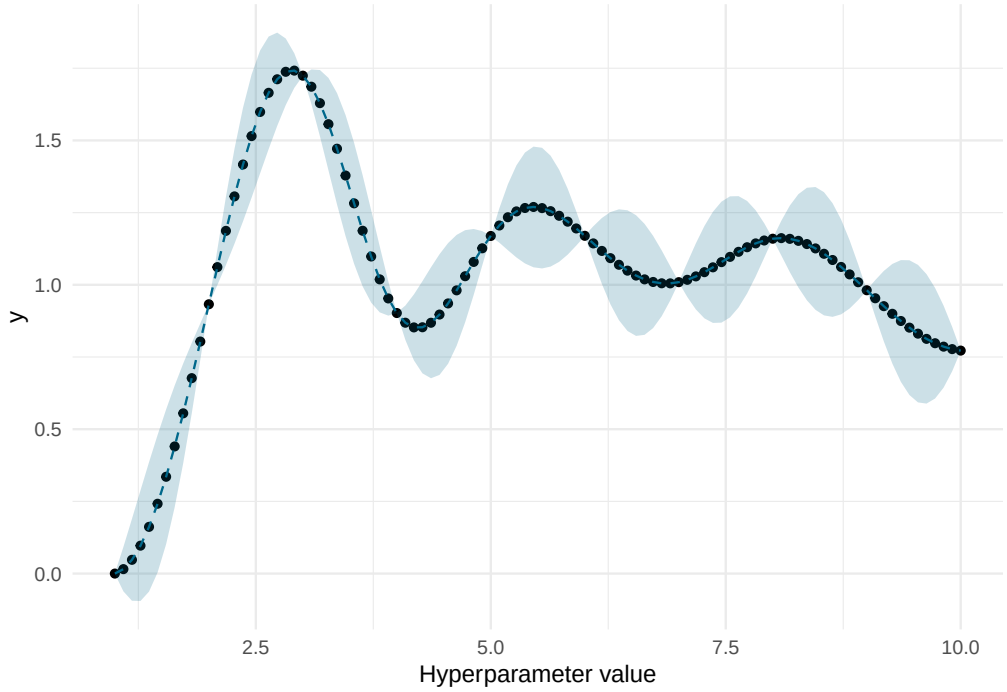


Figure 8: Examplary Gaussian Process for a single Hyperparameter

8 show an exemplary GP which was generated by predicting 100 possible candidates between 1 to 10 for the hyperparameter value x and evaluated them using an arbitrary evaluation function y, defined by equation 30.

$$y = \frac{a + (\sum_{i=1}^{k} b_i)^2 - 0.69 \times a^3 + log(x)}{2 \times a + log(x)} \tag{30}$$

with $a = \mathcal{U}(1,10)$ being a random variable drawn from the uniform distribution between 1 and 10 and $b = \mathcal{N}(0, 0.5^2)$ beeing a random variable drawn from the normal distribution with mean 0 and variance of 0.5^2. We observe that the confidence bound increases in size, the greater the distance between two candidates who have already been evaluated. Considering that our goal would be the hyperparameter value that minimizes the evaluation function value the GP would continue sampling values around $x = 4$ and probably between $x = 9$ and $x = 10$ since we would expect here the largest reduction in terms of the evaluation function given the estimated confidence bound.

The next step of the algorithm is to sample and evaluate new possible hyperparameter sets for the objective function. The GP helped to reduce the evaluation time as the process can estimate metric outcomes with some uncertainty. To take the decision which hyperparameter set should be evaluated next with the actual objective function, the Bayesian HPO algorithm uses an acquisition function. The output of this function told us which hyperparameter set x is the most promising to evaluate next with the objective function for the under-

lying model. Possible candidates for the acquisition functions are the lower (minimization)/upper (maximization) confidence bound or the probability of improvement and the expected improvement (EI) Wilson et al. (2018). The implementation of the Bayesian HPO used in this thesis was the EI. The EI is defined as

$$EI(\mathbf{x}) = \mathbb{E}[\max(0, f(\mathbf{x}) - f(\hat{\mathbf{x}}))] \tag{31}$$

where $f(x)$ is the GP estimation of our underlying model metric with hyperparameter set x, and $f(\hat{x})$ is the best performance that we have tried so far, since the GP is modelling a function of the probability of the model's metric dependent on the hyperparameter set x. For any set of X, we generate a normal distribution for the prediction of the value of f, such that :

$$p(f(\mathbf{x}) \mid \mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x})). \tag{32}$$

Following Brochu et al. (2010), one integrates the probability distribution from equation 32 multiplied with the difference of the performance from the hyperparameter set on trial and the best hyperparameter set so far. This leads to

$$\begin{aligned} EI(\mathbf{x}) &= \int_{f(\hat{\mathbf{x}})}^{\infty} p(f(\mathbf{x}) \mid \mathbf{x})(f(\mathbf{x}) - f(\hat{\mathbf{x}}))\, df(\mathbf{x}) \\ &= \int_{f(\hat{\mathbf{x}})}^{\infty} \mathcal{N}(f(\mathbf{x}) \mid \mu(\mathbf{x}), \sigma(\mathbf{x}))(f(\mathbf{x}) - f(\hat{\mathbf{x}}))\, df(\mathbf{x}) \\ &= (\mu(\mathbf{x}) - f(\hat{\mathbf{x}}))\Phi(z) + \sigma(\mathbf{x})\phi(z) \end{aligned} \tag{33}$$

with ϕ defining the probability density function of the standard normal, Φ being the cumulative distribution function of the standard normal distribution, while z is defined as the difference between the GP mean and the best performance so far scalled by the standard deviation of the GP. Formally

$$z = \frac{\mu(\mathbf{x}) - f(\hat{\mathbf{x}})}{\sigma(\mathbf{x})}. \tag{34}$$

The GP estimated $\mu(x)$ and $\sigma(x)$ that helped us to find the x, which we believe gives us the best EI for our next sample of x

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}_{t+1}} EI(\mathbf{x}_{t+1}). \tag{35}$$

Thereby, the Bayesian HPO creates a balance between exploring uncertain hyperparameters (exploration) and gathering observations from hyperparameters near the optimum (exploitation) Snoek et al. (2012). The algorithm stops after a user defined number of rounds without an positve expected improvement on the models metric.

4.7.5 Hyperparameter of tree ensemble models

In this subsection we will discuss the hyperparameter of the Random Forest, GBM, XGBoost, Light Gradient Boosting and CatBoost methods used in this thesis.

4.7.6 Random Forest

Random Forest is cited by several publications as a model which performs good with default parameters, i.e. without tuning Shahhosseini et al. (2019). Additionally, there are many hyperparameter to tune compared to gradient boosting based models like XGBoost. The main hyperparameters for a random forest model are *mtry* and *minimal node size*, which are defined ,in the algorithm used in this thesis, as *mtry* and min_n respectively.

4.7.7 Gradient Boosting methods

The gradient boosting method implementations of GBM,XGBoost,LightGBM and CatBoost used in this paper feature different hyperparameters to tune as shown in table 3. The *trees* hyperparameter is usually not tuned but fixed for a specific value, e.g. in Wang et al. (2020) and Karetnikov (2019).

4.8 Metrics

Metrics, in modeling applications, are functions to evaluate and compare the performance of models. Such metrics are also used as a loss function in gradient boosting methods to evaluate each (tree) model performance. A recent review study on machine learning results in team sports by Claudino et al. (2019) could not come up with a specific metric of choice but named those which achieved the best results in their sport. Tofallis (2015) mentioned as well that there is still not a specific metric of choice. Furthermore, they stress that there is no law for studies to just stick to one metric and added that each prediction method stores another piece of information. Therefore, this section

Table 3
Hyperparameter tuned for each method

Parameter	Model	Range
tree depth	All	[1,100]
mtry (see table notes)	All	[0,25]
learning rate	XGBoost, LightGBM, Catboost	[0,1]
loss reduction	XGBoost, LightGBM	[0,∞]
tree depth	XGBoost ,LightGBM, Catboost	[1,∞]
minimal node size	XGBoost, LightGBM, Catboost	[1,∞]

Note: The upper bound of the hyperparameter mtry differs due to the chosen method. This was up to 25 for Random Forest and XGBoost and Lightgbm models. Catboost models, which uses ordered target statistics to encode factor variables, had only up to 22 variables. The latter, as explained in the section 4.6, do not expand the dataset but encode the variable itself.

will look into several prediction metrics for model selection with respect to former sports and especially, road cycling prediction results and benefits for our chosen methods.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}} \tag{36}$$

where y_t is the is the t^{th} real observation of the dataset, $\hat{y}_t$ is the t^{th} predicted observation of the dataset and n is the total number of observations of the dataset. The RMSE is a more favorable metric when it comes to machine learning algorithm compared to of the MAE. This is due to the gradient of the RMSE of the t^{th} prediction which differs from that of the MSE.

$$\frac{\delta RMSE}{\delta y_n} = \frac{1}{2}\frac{1}{\sqrt{MSE}}\frac{\delta \text{MSE}}{\delta y_n} \tag{37}$$

In equation 37, we can see that gradient of the RMSE gives more weight to larger errors, which can be preferable if the dataset has many outliers.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\left|y_t - \hat{y}_t\right| \tag{38}$$

The mean average error (MAE) is a quite simple metric but faces two main drawbacks. First, since the MAE is measured on the same scale as the data, the error can be biased due to different scales of the dataset variables.

$$MAPE = \frac{1}{n} \frac{\sum_{t=1}^{n} |y_t - \hat{y}_t|}{y_t} \tag{39}$$

The mean absolute percentage error (MAPE), sometimes referred to as mean magnitude of relative error (MMRE), is widely used in businesses and organizations Tofallis (2015), as well as in football Strnad et al. (2015) and professional road cycling performance prediction Karetnikov (2019). Tofallis (2015) showed that, in regression tasks, the MAPE metric will tend to under-predict the forecast. The latter means that for the same error is higher when $\hat{y}_t < f_n$ compared $\hat{y}_t > f_n$. The advantage of this metric is the intuitive interpretation compared to the RMSE. That is because the latter presents us an absolute value which needs specific knowledge of the field to decide if the forecast can beat other forecasting methods such as an expert's guess

The coefficient of determination better known as R^2 is the relative amount of the variation in the target variable y that is predictable from the dependent variable matrix X with y a $n \times 1$ vector with n observations and X is a $n \times m$ matrix with m regressors. A regressor can be defined as an explanatory variable for the target variable y and can be either a variable of the dataset, a combination of variables or a scaled variant of a variables such as $log(variable)$ or $variable^2$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \hat{y})^2} \tag{40}$$

with SS_{res} = residual sum of squares and SS_{tot} = total sum of squares. The downside of the R^2 is that it always increases when more regressors are added to the model. This might not be meaningful metric since we can't compare models that feature a different number of variables.

A variant of the R^2 metric is the adjusted $\bar{R}^2$:

$$\bar{R}^2 = R^2 - \left(1 - R^2\right) \frac{m}{n - p} \tag{41}$$

with $p = m - 1$. By depending the R^2 on the amount of regressors m, we punish models for each regressor, so that models with a large amount of regressors do not automatically achieve a better metric value.

$$\bar{R}^2 = R^2 - \left(1 - R^2\right) \frac{k}{n - p} \tag{42}$$

Due to the different handling of factor variables by the tree-based ensemble methods the feature space can be different between methods. This is true at least when we compare a model fitted with XGBoost, which expands the feature space due to one-hot encoding, and CatBoost, which uses orderd target encoding and does not expand the feature space. In this comparison CatBoost ist always favored over XGBoost due to the factor variable handling. We have decided to use the $RMSE$ metric as the loss function for all gradient boosting methods and to compare all models and choose the best model based on the $RMSE$ metric as well. The R^2 metric of the best model, for the prediction of the average power measurement and the UCI weekly points, will be analyzed to give a more intuitive explanation of the prediction quality of the best model.

4.9 SHAP

A quite recent approach to interpret ensemble models was suggested by Lundberg and Lee (2017), arguing that SHAP values can be used to shed light on the inside of black-box models. The latter name is often used due to the non-parametric approach, in which we do not estimate a direct influence of a variable on the model, which leads to model results that can be used for prediction but without extensive knowledge on how the variables of the model influenced the model and the predictions. In this section, we will follow Lundberg and Lee (2017) to present a short formal explanation on how their approach works. SHAP values are based on Shapley values, first proposed by Shapley (1952), who used a method from the coalition game theory to predict the payout for each player who participates. Consider each feature of a dataset as a "player", the "game" as the prediction task of the model and the "payout" as the contribution of the feature to the prediction value. SHAP values predict feature importance in the same greedy fashion as ensemble methods do. The latter means that SHAP values are calculated by refitting the model for possible feature subsets $S \subseteq C$, where C is the set of all features, in an additive manner. This additive feature attribution method contains an explanation model g that is a linear function of binary variables such that

$$g\left(z'\right) = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{43}$$

$$\tag{44}$$

where $z' \in \{0,1\}^M$ is the coalition vector, M defines the maximum coalition size

and $\phi_j \in \mathbb{R}$ are the Shapley values.

The Shapley values ϕ_j in equation 44 are defined by

$$\phi_j(val) = \sum_{S \subseteq \{1,\dots,p\} \setminus \{j\}} \frac{|S|!(p-|S|-1)!}{p!} val(S \cup \{j\}) - val(S)) \tag{45}$$

where S is a subset of variables of the fitted model, x is the vector of variable values of the observation to be explained, and p the number of variables. $val_x(S)$ is the prediction for variable values in set S that are not in included in set S.

$$\text{val}_x(S) = \int \hat{f}\left(x_1, \dots, x_p\right) d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X)) \tag{46}$$

Performing multiple integrations for each feature that is not contained in the set S is possible but skipped in detail in this work. SHAP values satisfy three properties :

1) *Local accuracy* With $\phi_0 = E_X(\hat{f}(x))$, SHAP values have to be equal to the predictions of the model so that :

$$\hat{f}(x) = g(x')\phi_0 + \sum_{j=1}^{M} \phi_j x'_j = E_X(\hat{f}(X)) + \sum_{j=1}^{M} \phi_j x'_j \tag{47}$$

2) *Missingness* The property of missingness states that $x'_j = 0 \Rightarrow \phi_j = 0$ has to be valid. This property is needed to ensure that a constant features with some zero entries, which is part of the set S, always receives a Shapley value of 0. This requirement closes a possible loop-hole which could lead to a contradiction of property 1). The latter would be the case, if such a zero-entry would receive an arbitrary Shapley value.

3) Cosistency Assume that $\hat{f}_x(z') = \hat{f}_x(h_x(z'))$ and $z'_{\setminus j}$ implies that $z'_j = 0$. For any two models f and f' that satisfy

$$\hat{f}'_x\left(z'\right) - \hat{f}'_x\left(z'_{\setminus j}\right) \geq \hat{f}_x\left(z'\right) - \hat{f}_x\left(z'_{\setminus j}\right) \tag{48}$$

for all inputs $z' \in [0,1]^M$, then :

$$\phi_j\left(\hat{f}', x\right) \geq \phi_j(\hat{f}, x) \tag{49}$$

This property can be summarized as follows : If a model changes in such a way that the marginal increase or indifference of a feature value contribution is observed, c.t., the Shapley value also increases or stays the same.

For tree-based ensemble models, Lundberg et al. (2018) came up with a variant of SHAP, namely TreeSHAP. The latter method uses the conditional expectation $E_{X_S|X_C}\left(\hat{f}(x) \mid x_S\right)$ as the value function to calculate the Shapley values. For the expected prediction of a single tree, consider an observation x and a feature subset S from all the set, of all features C. If $S = C$, the expected prediction would be equal to the prediction from the node in which the observation x falls. If $S = \emptyset$ we would use the weighted mean of all predictions of all terminal nodes. If $S \subseteq C$, we only consider those predictions of terminal nodes that do not come from a path that contains a splitting rule dependent on feature $\overline{S}$, a feature that is not part of the feature subset S.

4.10 Models

This thesis aims to predict the average power of training sessions and the weekly UCI score of professional road cyclists from teams of the year 2020. For the prediction of the average power the Strava dataset came with some problems as described in section 2.2. We explained why we should impute values or discard observations with an average power of 100 or below and proposed replacement strategies like combining other average power variables to a single one or using an imputation method called IRMI to calculate replacement values for those target variable values in concern. Discarding or imputing *NA* values in two of the predictor variables $avg\_calories$ and $avg\_temperature$ were used by the author to observe if a model with imputed values ,or at least without discarding observations, can improve predictability of the target variable. Therefore, we came up with 4 models for the prediction of the average power and for the prediction of the UCI_weekly_score. Furthermore, the values of all observations of all models were normalized. Numerically one should not expect a difference between scaled and unscaled regression using tree-based methods. Instead, we anticipated that the speed of the training process of those models will be increased due to faster converging, especially gradient descent methods. For the SHAP analysis we decided to not scale the test dataset to accomplish a more intuitive analyzation of the best model.

Table 4 gives an overview which model is used in combination with the three target variables and shows how the Strava dataset of each model differ, in terms

Table 4
Model presentation for the Strava dataset

Target Variable	Model 1 (IRMI and comb. avg. power) (61,840)	Model 2 (Dropped NA and comb. avg. power) (54,960)	Model 3 (IRMI and avg. power) (61,840)	Model 4 (Dropped NA and avg. power) (51,440)
Comb. avg. power	✓	✓	✓	✓
Avg. Power	✓	✓	✓	✓
UCI weekly points	✓	✓	✓	✓

Note. Below of each model name the sample size of the Strava dataset used in this model is given in parenthesis. Each model uses all variables from 1 with a *.

of observation size. All models use 75% of the data as training and 25% for testing. Train and test sampling was random when we predicted the average power.

Since the UCI weekly points variable is updated for each athlete every week, we would probably leak some information about the values in the beginning of the distribution of the UCI weekly points variable if we were to sample training and test observations randomly as well. Therefore we decided to use the first 75% of the observations for training and the last 25% for testing. For the prediction of the average power variables we will use two out-of-sample testing strategies for all models. The 5-fold cross-validation uses $\frac{4}{5}$ for training and $\frac{1}{5}$ for validation from the training sample of the Strava dataset. The best model will then be used to predict on the test samples from the Strava dataset. We call the latter procedure out-of-sample (OOS) testing. For the UCI weekly points variable 5-fold CV would also lead to possible leakage of information, since those folds with observations from the end would contain information about observations of the beginning of the variable, so that we will only use the OOS testing strategy to evaluate the different models. Furthermore, if we use train dataset and test dataset in the upcoming results we refer to the train and test samples from the Strava dataset in described in this section.

To analyze how good gradient-boosting techniques perform against a less complex regression method such as Random Forest, we will use the Random Forest method as our benchmark method. Furthermore, we will compare all of our model performances to the performance of model 4. The latter is the model with the most conservative NA value handling since all values of avg_power below 100 and all NA values of the variables $avg\_calories$ and $avg\_temperature$ were discarded.

5 Results

This section will present and explain the results gained from the methods and models presented in the former section. An overview of the 5-fold cross-validated results of the training dataset of all methods is presented in table 5. Afterwards, we will analyze how the out-of-sample (OOS) results of each method differs with the default hyperparameter sets and the hyperparameter sets obtained from the Bayesian optimization process. The best model and method combination is then explained with a SHAP analysis. The latter method will help to understand which variables improved the model in terms of contribution to the prediction values. The last subsection will take a look on the different hyperparameter settings of the methods and show if we can define a possible range of those tuned hyperparameter settings that improved a model the most.

5.1 Prediction of the average power

Table 5
5-fold CV RMSE results for all models predicting the average power

Method	Model 1	Model 2	Model 3	Model 4
		Default Hyperparameters		
RF	0.359 (0.00358)	0.349 (0.00353)	0.345 (0.00373)	0.321 (0.00301)
LGBM	0.923 (0.00913)	0.915 (0.00858)	0.92 (0.00754)	0.914 (0.00481)
XGBoost	0.395 (0.00444)	0.382 (0.00537)	0.34 (0.00226)	0.337 (0.00497)
Catboost	1 (0.00947)	1 (0.00848)	1 (0.00722)	1 (0.00423)
		Tuned Hyperparameters		
RF	0.316 (0.00402)	0.317 (0.00545)	0.274 (0.00392)	0.269 (0.00299)
LGBM	0.224 (0.00414)	0.25 (0.0044)	0.181 (0.00282)	0.196 (0.00596)
XGBoost	0.23 (0.00322)	**0.211 (0.00389)**	**0.148 (0.00274)**	0.165 (0.00443)
Catboost	**0.21 (0.00553)**	0.233 (0.00561)	0.184 (0.00372)	**0.159 (0.00543)**

Note. Each RMSE metric is 5-fold cross validated. Standard deviations are given in the parenthesis. Bold numbers indicate the lowest RMSE for each model. Hyperparameter tuning was performed with Bayesian HPO for all methods.

Table 5 shows the IS results for all models as described in the section 4.10 and all methods described in 4. For the results with the default hyperparameter of each method, we have observed no method that achieved the lowest RMSE for a model compared to the methods with tuned hyperparameter settings. The lowest RMSE metric for methods with default hyperparameter was observed with RF and the model 4 with an RMSE value of 0.321. With default hyperparameter settings the Random Forest method has beaten all other models in terms of the lowest RMSE value. XGBoost seemed to perform with default hyperparameter settings comparably good as Random Forest, while the other two methods

failed to do so. LGBM and Catboost tend to achieve RMSE values which are up to 3 times higher than the results of the Random Forest and XGBoost method in the default hyperparameter settings. Catboost performed the worst for all models with an RMSE equal to 1 for all models. The opposite was observed for the model results with tuned hyperparameter settings using Bayesian HPO. Catboost outperformed all other methods for model 1&4 and achieved the second lowest RMSE overall with 0.159 for model 4. In summary, we observed an error reduction more than 7 times lower compared to the Catboost results with default settings. LGBM improved its performance in similar levels than, Catboost but failed to outperform any other model with and without hyperparameter tuning. The lowest RMSE value of 0.148 was achieved by XGBoost with model 3 and tuned hyperparameter settings. The reduction of the RMSE value for XGBoost with tuned hyperparameter settings was up to halve compared to the results from the default HP section. The improvement of the RMSE value of the Random Forest method with tuned hyperparameter settings compared to their RMSE results with default settings was low, with a maximum reduction of the RMSE of 0.047. In both hyperparameter sections of the table 5 we observed that the model 3 leads to the lowest error metric. Comparing the results model-wise, we discovered that, independent from the hyperparameter aspect, the models 3 & 4, which predicted the avg_power variable, achieve in total better results than model 1 & 2 which predicted $avg\_power\_combined$, a variable combined from original measurements by the bicycle computer of the athlete, and some estimations from the Strava website. Moreover, model 3 seems to beats model 4 with most methods. Possible explanations could be that $avg\_calories$ and $avg\_temperature$ had a large impact on the prediction of the avg_power variable and/or the observations that did not have to be discarded by imputation provided significant value to the predictions. Some of these suggestions will be adressed in the forthcoming SHAP value analysis.

We will continue our analysis of the results comparing the OOS results for model 3 with respect to the default and tuned hyperparameter settings.

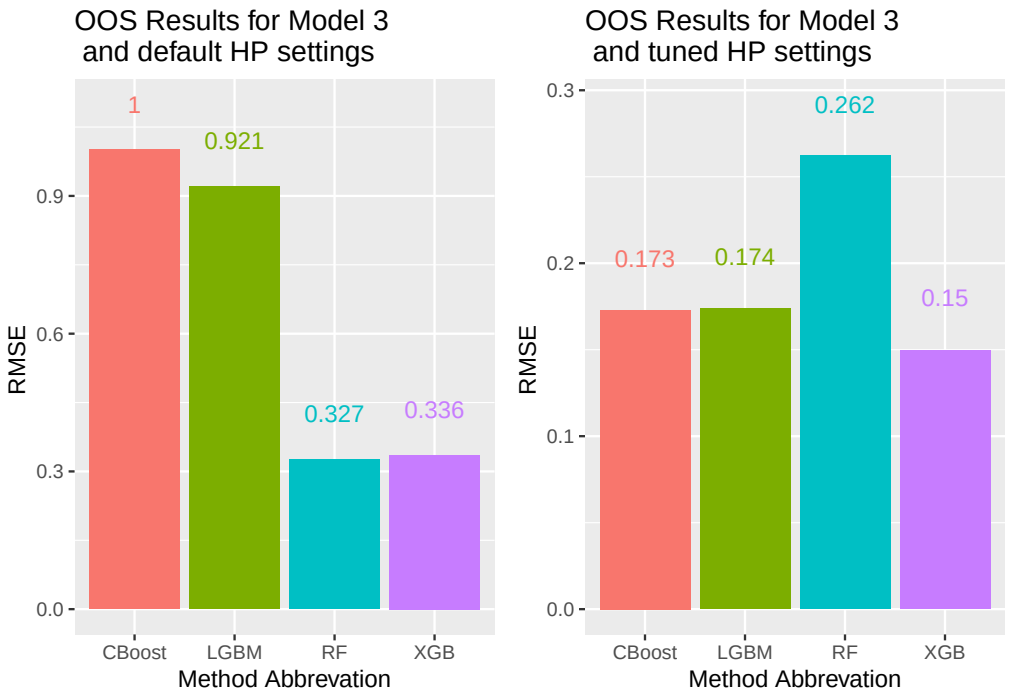


Figure 9: OOS results for model 3

For model 3 the OOS results were highlighted in figure 9. When we discussed the IS results of table 5, we argued that model 3 achieved the lowest RMSE score of all models with the XGBoost method and tuned hyperparameter settings. Related results were observed for the OOS calculation of the RMSE metric. Catboost and LGBM failed to achieve comparably low results with default hyperparameter settings while the Random Forest and XGBoost method achieve equally low RMSE values of 0.327 and 0.336. When we used tuned hyperparameter settings to train and test model 3, we observed an error reduction almost up to 8 times for Catboost compared to the OOS results with default hyperparameter settings. With an RMSE of 0.15, the XGBoost method achieved the lowest RMSE value, as in the IS scenario. The Lightgbm and CatBoost methods came in second with equally low but slightly higher, compared to the XGBoost method, RMSE values of 0.921 and 1.001 respectively. The R^2 of the model 3 fitted with the XGBoost method is 0.9776897. This implies that more than 97% of the variance of the prediction values can be explained by the model. This is a very high value for any model, and we will limit this result in the discussion section.

As mentioned before we will further explain the best model, model 3, trained by the XGBoost method, in detail using SHAP. The Shapley values of SHAP are stored in a matrix so we can analyze either the whole model or specific observations. The SHAP feature importance plot presents features of the model ranked by their Shapley values in a decreasing manner. This means that the

variables on top have a larger influence than those at the end. Whole model importance is achieved by the average of the absolute Shapley values per feature I_j of the whole test dataset.

$$I_j = \frac{1}{n} \sum_{i=1}^{n} \left| \phi_j^{(i)} \right| \tag{50}$$

with n equals the number of observation of the test dataset, and i equals the i^{th} observation of the test dataset.

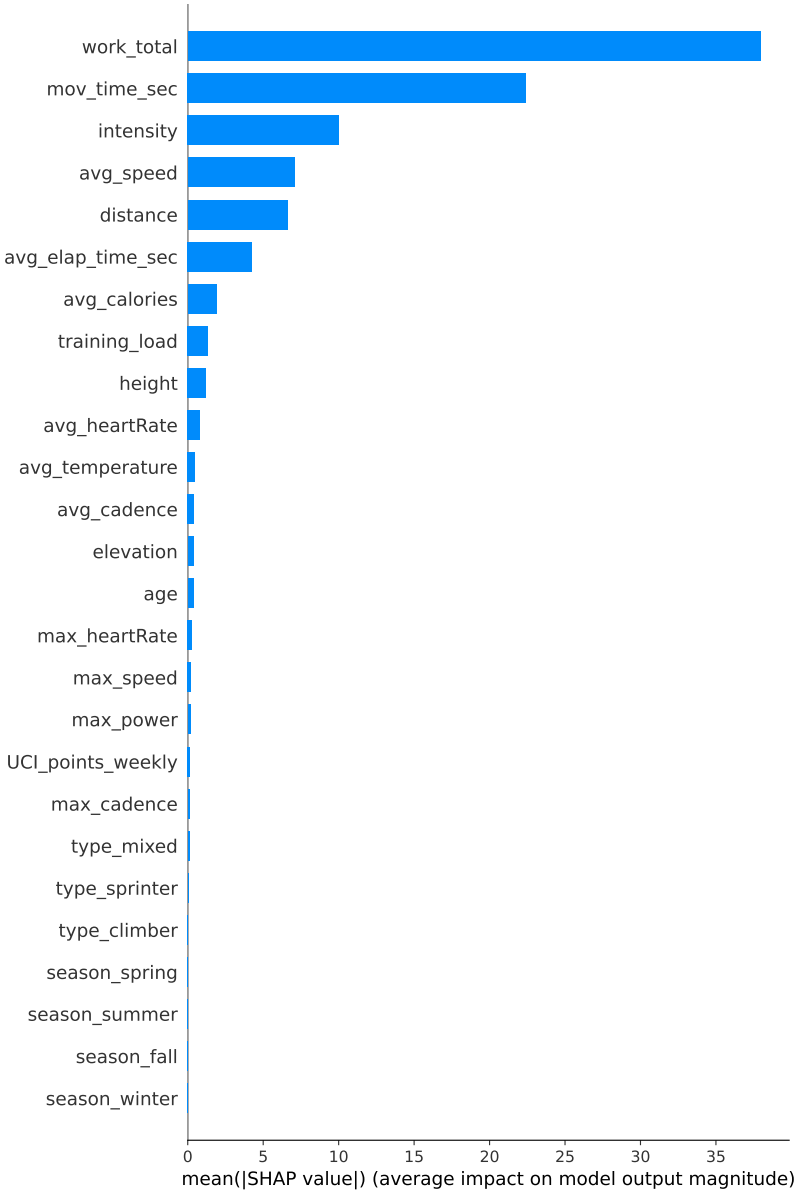


Figure 10: This figure shows the feature importance defined by the absolute mean Shapley values of each variable of model 3 fitted with the XGBoost algorithm. All values were calculated from the test dataset. For

Figure 10 shows that the feature $work\_total$ on (absolute) average added around 40 watts to the mean prediction value of the target variable avg_power,

while mov_time_sec is predicted to have the second largest impact on the prediction. If the latter feature is part of the model the average increase to the average prediction values around 20 watts. On the other hand variables with almost any contribution to the average prediction are UCI_points_weekly and $max\_cadence$, $type$ and $season$. It seems that the weekly tournament points, the maximal cadence value of the training ride and the cycling type of an athlete are not helpful to predict the average power of a training ride of a professional athlete measured through a bicycle computer. The categorical variable $season$ seems to have no impact at all on the model. We can therefore conclude that we do not observe any seasonal influence on the average power of a training ride. All effects describe the behavior of the model and are not necessarily causal in the real world. Furthermore we do not claim that it would not matter if a road was covered with ice, nor do we claim that snow in the winter month would have no impact on the average power of a training ride compared to a training on the same road in a summer month. A reasonable explanation of these results could be, that those professional riders, which faces seasonal influence on their training routes, might train during this time of the year in the southern hemisphere or use a static bicycle trainer at home.

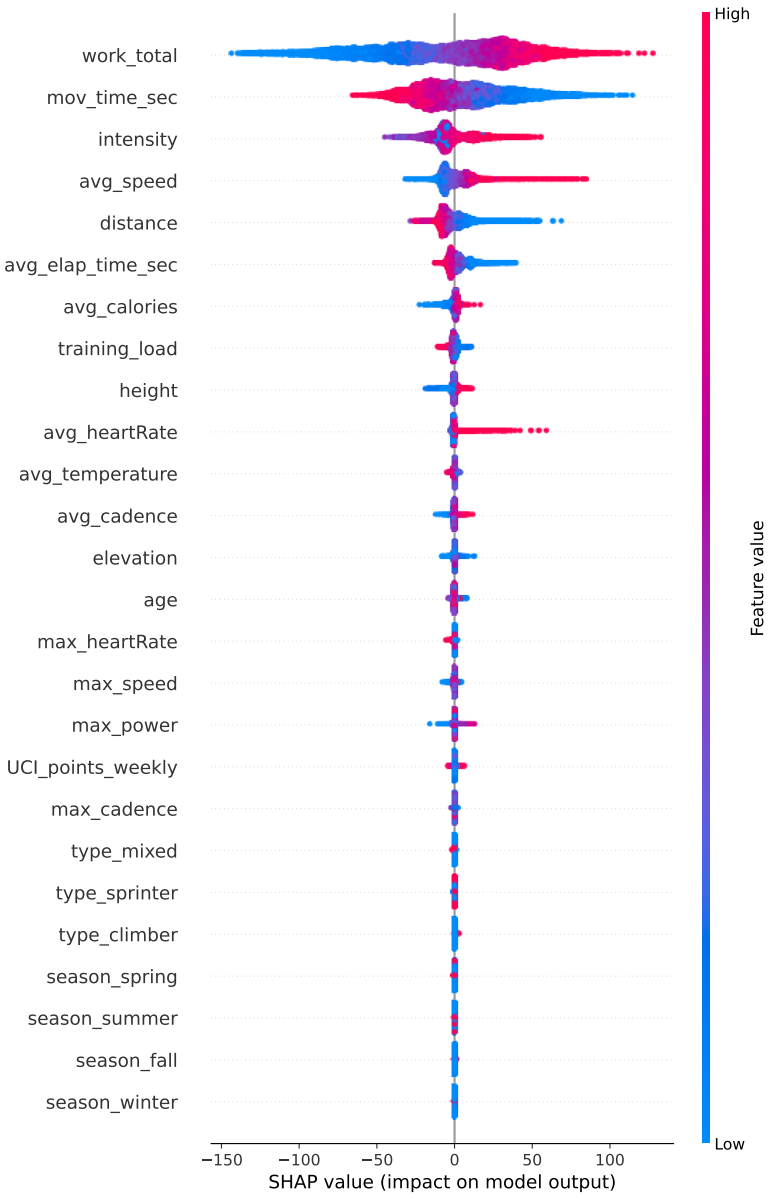


Figure 11: This figure shows the summary of each variable with respect to each observation of model 3 fitted with the Catboost algorithm. All values were calculated from the test dataset.

The SHAP summary plot in figure 11 splits the feature importance bar of each feature in figure 10 into their single observations and adds a heat map to reflect the feature value. More specifically, bluer dots imply a lower and redder spots a higher feature value. Grey values mark features that are categorical. For model 3, we observed that higher $work\_total$ feature values led to a larger increase of the average prediction value of the avg_power variable. We observed that most observations helped the model prediction by increasing or decreasing the average prediction with values between ∓ 50. With a slightly higher accumulation of observations around +50, we expected that $work\_total$ increased the average prediction value of the model more often than it decreases it. This

helped us describe the average influence of $work\_total$ as likely positive, which we were unable to tell just with the bar plot due to the absolute values. mov_time_sec feature values show an opposite effect on the average prediction of avg_power. Higher values tend to decrease the model's prediction while lower tend to increase it. This effect is in line with the intuitive judgement that a training ride shorter in time should be on average less powerful than one that requires more time. The same behavior is expected from the $distance$ feature. A training ride that is longer in distance should decrease the average prediction of the model, as we would expect factors such as exhaustion, to reduce the overall average power that can be used during the training ride. Biological features such as $height$ or age seem to have no large impact on the model. Taller riders show an increase of the average model prediction of the average power.Some younger professionals show a small benefit from their age in terms of a surplus of the model's prediction value, while professionals with a higher age show neither a decrease nor an increase of the model's prediction due to their age. The latter could indicate that some younger professionals might have a biological benefit while older professionals benefit from their experience gained over their career years.

5.2 Prediction of the UCI weekly points

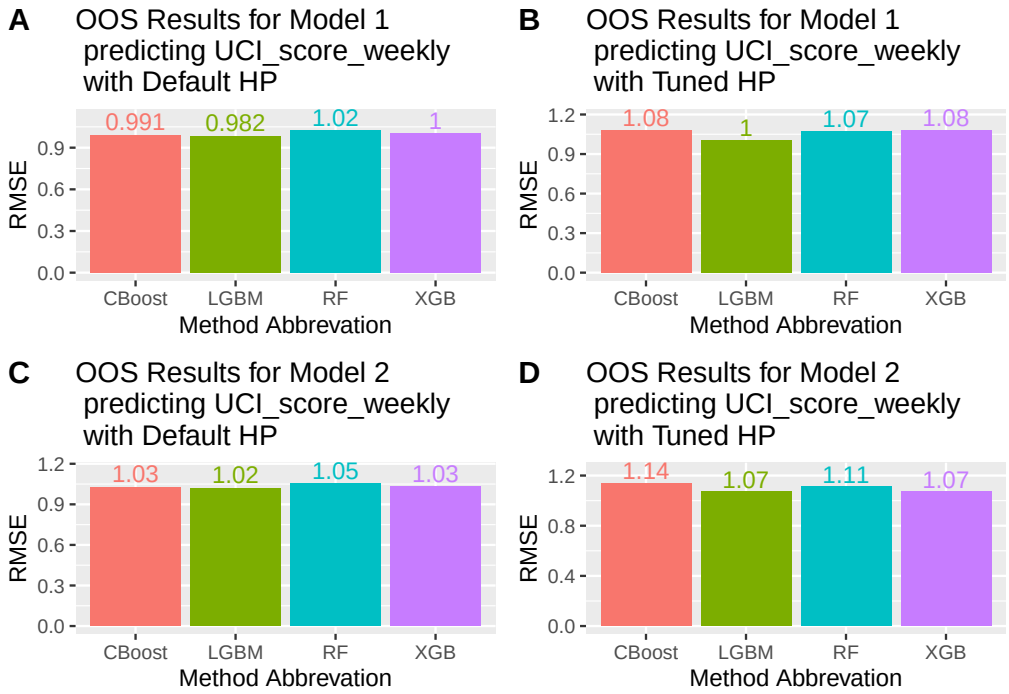


Figure 12: OOS results for UCI weekly points prediction with model 1 and 2

Figure 12 shows the OOS results for model 1 (A and B) and 2 (C and D) predicting the UCI weekly points. The lowest OOS RMSE value model-wise is 0.982396 given by model 1 fitted with the Lightgbm method. The worst results in terms of performance for model 1 targeting the weekly tournament points is Catboost in the tuned hyperparameter settings with an RMSE of 1.14. As already mentioned we observe a performance decrease over all methods when we compare default to tuned hyperparameter settings. A notable result is that Catboost, which was always in the top 2 method results for fitting a model, fails to repeat those results for the UCI weekly points prediction with model 2. Catboost managed to achieve the worst results in the tuned hyperparameter settings with an RMSE of 1.14. A possible explanation for the worse results from methods with tuned hyperparameter settings could be that we stopped the optimization process for all models if no RMSE improvmen was observed for 3 consecutive rounds with the Bayesian HPO. Furthermore we expect that the default settings might be near the optimal settings for this regression task and the given Strava dataset. The results are in line with our expectation from figure 2.

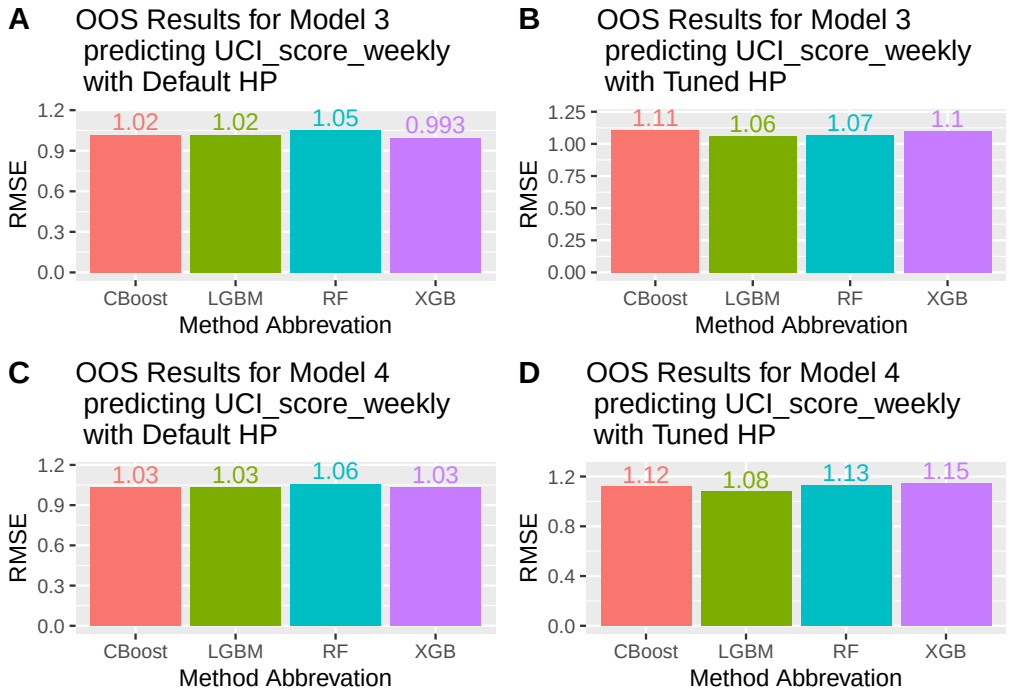


Figure 13: OOS results for UCI weekly points prediction with model 3 and 4

Figure 13 presents the OOS results for model 3 (A and B) and 4 (C and D) predicting the UCI weekly points. Compared to the OOS results of all other model results, model 3 managed to produce the second best performance with default hyperparameter settings. The XGBoost method accomplished an RMSE value of 0.993. For model 4 predicting the UCI weekly points, XGBoost showed the worst performance result in the OOS and tuned hyperparameter settings

with an RMSE of 1.15. For the default settings Lightgbm again achieved a little lower RMSE value of 1.02 with default settings compared to all other methods.

No model-method combination managed to achieve a RMSE close to prediction of the *avg_power* variable. Furthermore, we observed that for every model the OOS results of the methods with default hyperparameter settings achieved better results compared to those with tuned hyperparameter settings. These results were not useful for predicting professional rider ranks but were to some extend in line with other findings such as Brefeld et al. (2020). The previous authors managed to predict only 5 of the 10 ranks correctly for the Tour of Flanders in 2019. They argued that team sports might have a huge unpredictability factor, which explained the poor performance to some extend. Nevertheless, the results for the prediction of the UCI weekly points in this work are much worse then their results. We will continue with the SHAP value analysis to shed light on possible features which led to the poor model performance.

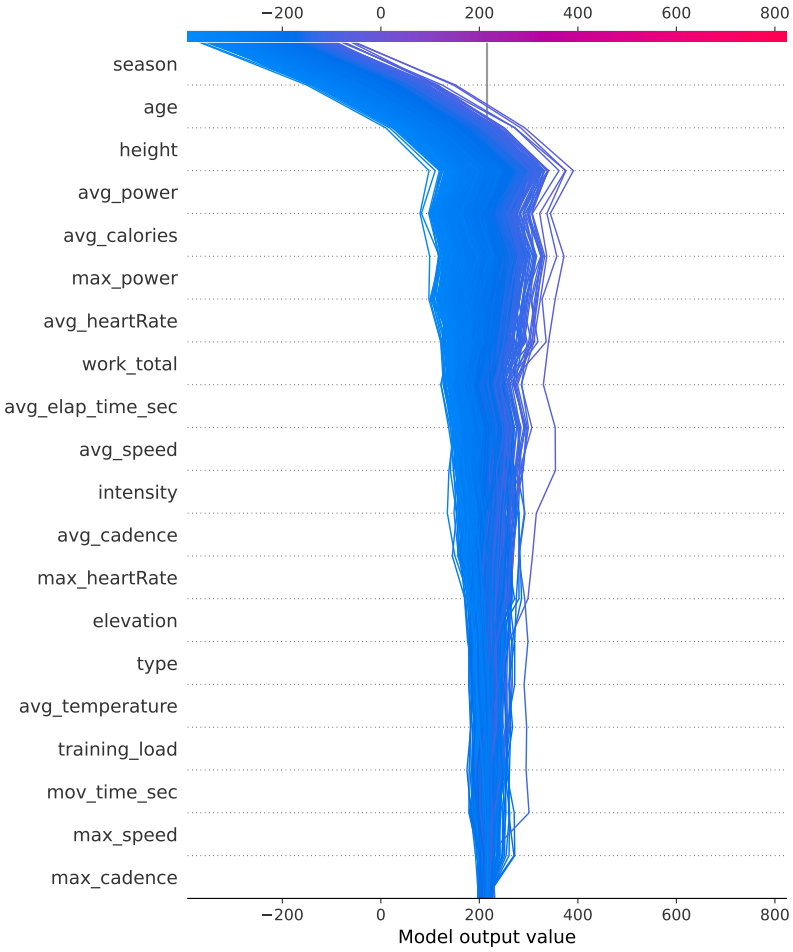


Figure 14: This figure shows a feature decision plot of each variable of model 1 fitted with the Lightgbm algorithm predicting the UCI weekly points. All values were calculated from the first 10.000 observations the test dataset.

We will begin the SHAP analysis of model 1 which has predicted the UCI weekly points with the analyzation of figure 14. This figure shows the impact of each feature on the model prediction of the first 10.000 observations. The horizontal grey line around the value of 200 indicates the expected model prediction without any feature added. A heat map colors each observation indicating the difference from the expected model prediction. Bluer lines imply a lower and redder lines a higher outcome of the observation. Almost all values seem to be underfitted by the model. The actual range of the UCI weekly points in the first 10.000 observations of the test dataset is between [0;1944] and an average value of 224.3048. the biggest influence on the negative divergence from the mean prediction of the model are the variables $sesaon$, age and $height$. The variable with the forth highest influence is avg_power which show a small positive shift towards values above the mean prediction of the model.

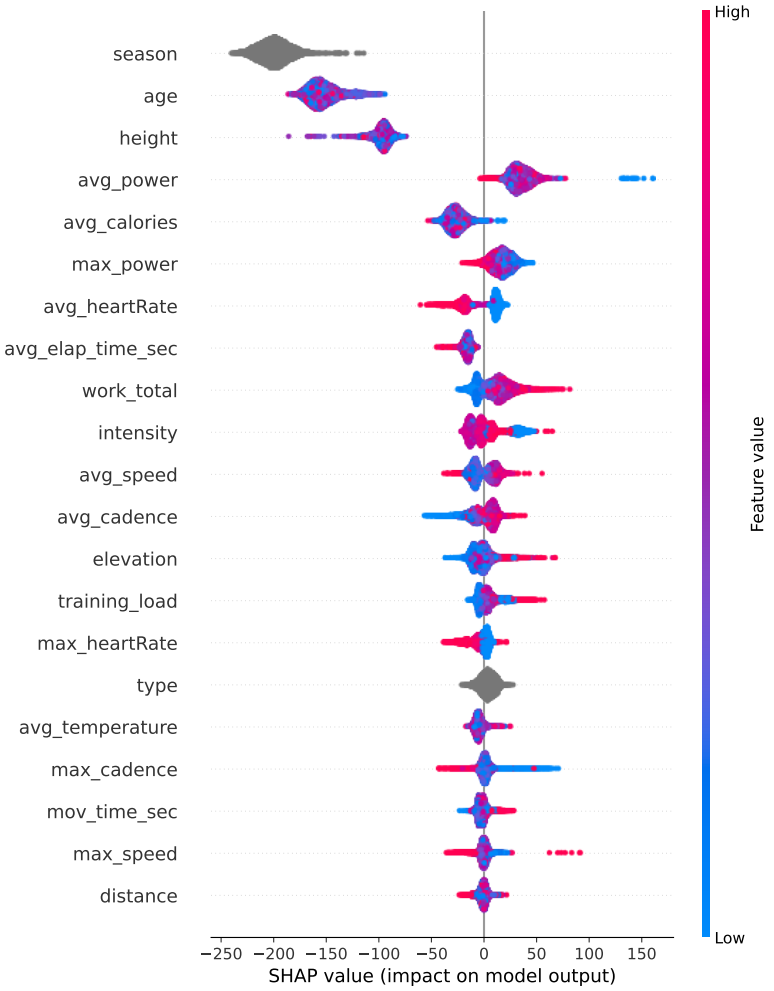


Figure 15: This figure shows the summary of each variable with respect to each observation of model 1 fitted with the Lightgbm algorithm predicting the UCI weekly points. All values were calculated from the test dataset.

Figure 15 shows the summary of each feature on a heat map for the feature

values of all observations. We observe no heat pattern for *sesaon*, *age* or *height*. This means that higher/lower feature values do not indicate an increase/decrease of the model predictions. As expected from figure 14, we observe that all values of those variables show a negative impact. The difference between those three variables is the average impact indicated by clusters of the observations. For *age*, we observe a cluster around -150, while *height* shows a cluster around -100. The negative impact on the average prediction of the UCI weekly points of *age* is therefore on average around 1.5 times higher than the impact of *height* on the average prediction of the UCI weekly points. A cluster with an average positive impact of around 30 is observed for the *avg_power* feature. Some very low values of that feature seem to lead to an larger increase of the UCI weekly points prediction compared to the mean model output while very high values show an opposite effect. An explanation for this behavior might be that athletes who had a hard training session in the same week of some tournament might either not participate in the event or could not use their full capabilities due to their heavy exercise. Such an explanation could be supported when the feature *avg_heartRate* is examined. A higher average heart rate seems to lead to a lower UCI weekly points, while the reversed effect can be expected for lower average heart rate values on the training rides.

When we removed the variables *height*, *season*, *age*, *avg_calories*, *avg_elap_time_sec* from the dataset and refit all models again, we saw a very small performance increase for all models. The best model is still model 1 with the default hyperparameter settings for XGBoost. The latter achieved an RMSE of 0.961 an error reduction of 0.03 compared to Lightgbm with the same model.

Figure 16 presents the results for model 1 (A and B) and model 2 (B and C) while figure 17, presents the results for model 3 (A and B) and model 4 (C and D) of the prediction of the *UCI_score_weekly* variable without the variables *height*, *season*, *age*, *avg_calories*, *avg_elap_time_sec*. We observed a small improvement for all models with XGBoost being the method which achieves the lowest RMSE of 0.961 with model 1 and default hyperparameter settings. (A).

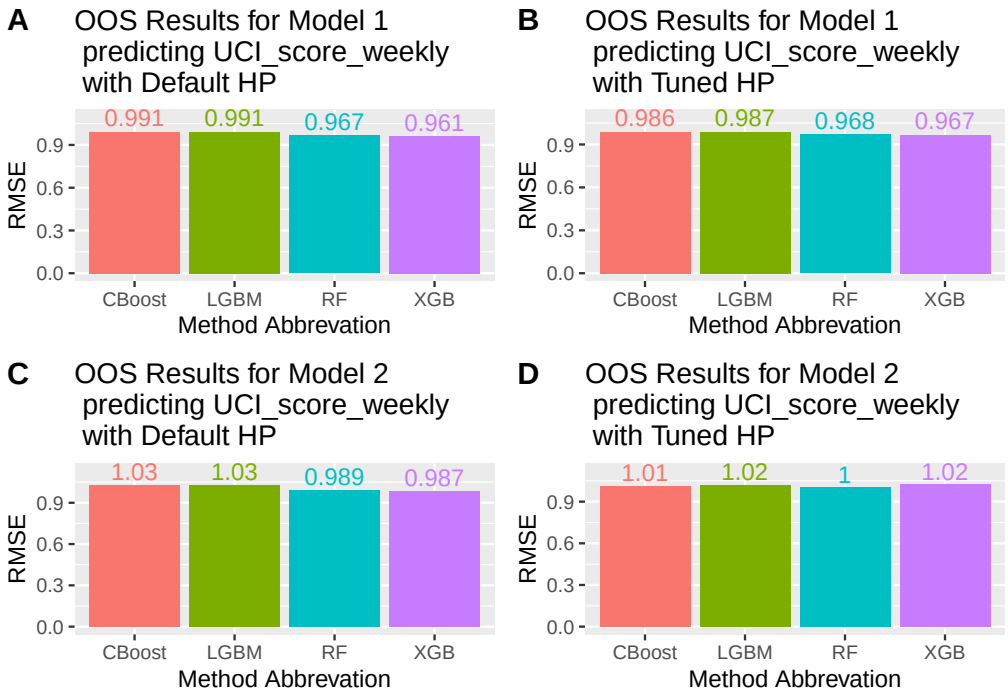


Figure 16: results for UCI weekly points prediction with model 1 and 2 without *height*, *season*, *age*, *avg_calories* and *avg_elap_time_sec*

We observed, that when we excluded those variable that had a negative impact , CatBoost and Lightgbm achieved a small $RMSE$ decrease when they were used to fit the models 1,2 and 3 with tuned hyperparameter settings. This helps our argument that Bayesian HPO improves the predicition results for tree-based methods, even if model 1 again shows the best result when XGBoost was used with default hyperparameter settings.

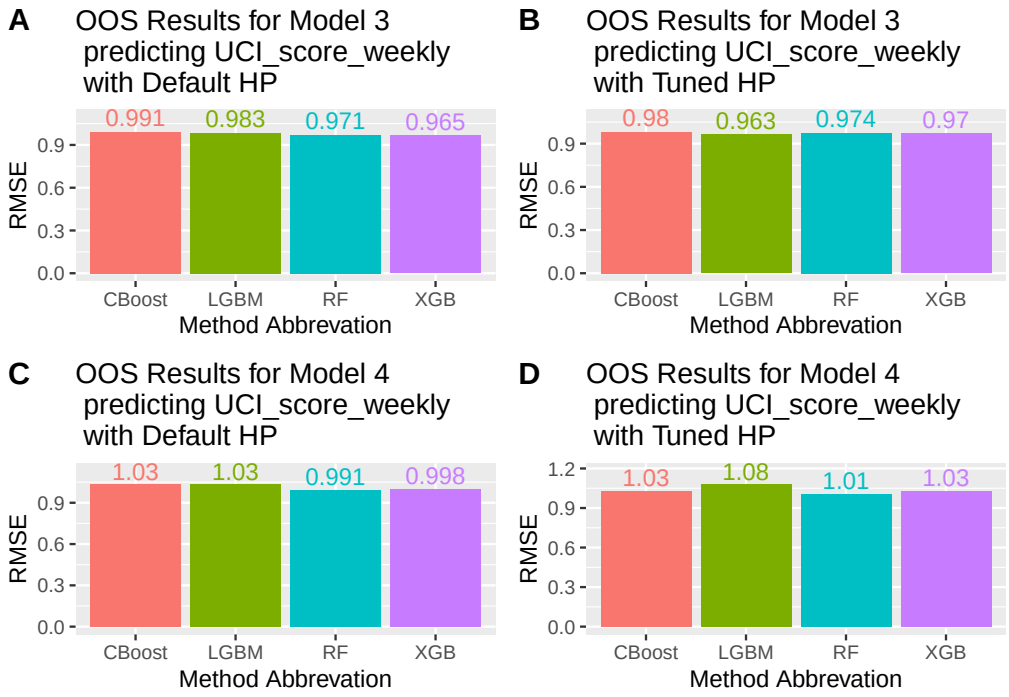


Figure 17: results for UCI weekly points prediction with model 3 and 3 without *height*, *season*, *age*, *avg_calories* and *avg_elap_time_sec*

5.3 Bayesian Hyperparameter Tuning

In the case of the prediction of the average power measurements, Catboost and Lightgbm seemed to rely heavily on HPO. Random Forest and XGBoost achieved with default hyperparameter settings equally good results. XGBoost benefited more from the hyperparameter tuning in terms of RMSE reduction than Random Forest. One reason for this result might be that the Random Forest method only has 2 hyperparameter to tune in comparison to gradient boosting methods which have up to 5 hyperparameter. Furthermore those extra hyperparameter could be more valuable in terms of RMSE reduction.

Table 6 shows the results of the Bayesian HPO and the default values of the best model with the method that achieved the lowest RMSE for both regression tasks. An overview over all methods for both best models can be found in the appendix figure 9

We detected that for the Strava dataset a combination of 20 features with at least 40 observations in a node to continue splitting, a maximum tree depth of 14, a learning rate of 0.0729534 and almost 0 but still positive loss reduction of 4.7306132×10^{-4} led to the best OOS RMSE results of model 3 predicting the average power of the training sessions. For the prediction of the UCI weekly points, we observed that fitting model 1 with XGBoost using the default hyperparameter settings described in table 9 section Model 1 UCI weekly points

Table 6
Hyperparameter Results of the Bayesian HPO of the best model

Method	mtry	Minimal node size	Tree Depth	learning rate	loss reduction
			Model 3 Average Power		
XGBoost	**20** (all)	**40** (1)	**14** (6)	**0.0729534** (0.3)	$\mathbf{4.7306132 \times 10^{-4}}$ (0)
			Model 1 UCI weekly points		
XGBoost	9 **(all)**	3 **(1)**	14 **(6)**	0.020941 **(0.03)**	5.0319939×10^{-5} **(0)**

Note. Values in square brackets indicate the default hyperparameter settings. Bold numbers indicate the hyperparameter settings of the method which achieved the lowest RMSE for the best model. '-' indicates that this hyperparameter is not available for tuning. The stop condition of the HPO was 3 consecutive rounds without an RMSE improvement of the model. All methods generated 1000 trees. *x indicates the number of variables of the train dataset.

led to the best OOS RMSE results.

6 Discussion

There is a high correlation between avg_power and $work\_load$ which is likely due to the same information, power expressed in watts, was used to calculate both variables by the bicycle computer. This might have lead to some form of target leakage in the sense that some information about the average power is already contained in the $work\_load$. We considered this and used the model with the highest prediction power in terms of the lowest OOS RMSE value to predict avg_power again without $work\_load$ in the Strava dataset.

The results are presented in table 7. We observe a comparable pattern to the OOS results of the models containing the variable $work\_load$. The RMSE is always lower when the method used tuned hyperparameter settings compared to the default settings for all models. In contrast we observe that either Catboost or XGBoost are the best performing methods for models 1 and 2,3,4, respectively. With model 3 now achieving the lowest RMSE of 0.297 which equals to an R^2 metric value of 0.908. So that even without the $work\_load$ variable we can explain more than 90% of the variance of the predicted average power.

Amoukou et al. (2021) claimed that SHAP values for tree ensembles might not be exact especially if the model had some one-hot encoded categorical variables. They argued that this happens due to the break of a fundamental

Table 7
OOS RMSE results for all models predicting the average power without $work\_load$

Method	Model 1	Model 2	Model 3	Model 4
Default Hyperparameters				
RF	0.434	0.412	0.388	0.404
LGBM	0.866	0.851	0.866	0.861
XGBoost	0.478	0.459	0.432	0.449
Catboost	0.996	0.991	1.001	1.003
Tuned Hyperparameters				
RF	0.42	0.394	0.386	0.392
LGBM	0.335	0.319	0.293	0.32
XGBoost	0.35	**0.316**	**0.297**	**0.305**
Catboost	**0.33**	0.344	0.321	0.309

Note. Bold numbers indicate the lowest RMSE for each model. Hyperparameter tuning was performed with Bayesian HPO for all methods.

property of SHAP values, namely the symmetry property or equal treatment of equals.

The Strava dataset had two categorical variables namely $season$ and $type$. $Cramesr's V$ is a correlation measure between two nominal variables.

$$\text{Cramer's } V = \sqrt{\left(X^2/n\right)/\min(c-1, r-1)} \tag{51}$$

where X^2 defines the chi-square statistic, n the number of observations, r the number of rows and c the number of columns. The range is between 0 and 1. The $Cramesr's V$ between $season$ and $type$ is 0.1247. Therefore we observe a low but positive correlation between these two variables. A simple example shows how this leads to biased SHAP values. Consider $Y = 100 \times X_1 + 100 \times X_1$ be a simple model. With X_1 and X_2 were both Bernoulli variables of parameter $p_1 = p_2 = 0.5$ and 1000 observations.

Table 8
Repartition of X_1 and X_2

Observations	X_1	X_2
400	**0**	**0**
100	**1**	**0**
100	**0**	**1**
400	**1**	**1**

Table 8 represents the repartition of the the latter variables : Since this is

symmetrical in X_1 and X_2 we should observe equal SHAP values without considering the correlation between these two variables.

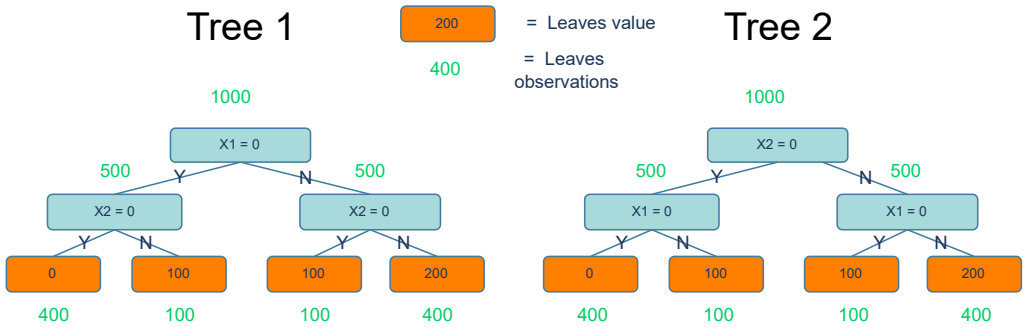


Figure 18: Diagram of the two tree versions of the simple model $Y = 100 \times X_1 + 100 \times X_1$. The values with a green number representing the subset of the dataset at a given split while the numbers in the leave nodes representing the value of the leaves.

Following Amoukou et al. (2021), table 10 shows the calculation of the SHAPE values of the simple model given that X_1 and X_2 share a positive correlation. The latter is represented by two trees of depth 2, each tree represented one version of the model depending on the order or X_1 and X_2. A visualization of of the aforementioned tree representation of the bias is given in figure 18. We also calculated the bias of this example. Table 10 in the appendix shows that the SHAP values are not symmetric. We follow Amoukou et al. (2021) and assume that in general SHAP values for tree ensembles are biased if two categorical variables in a dataset share a positive correlation and were used at least once as a splitting criterion. Therefore we assume that the SHAP values for the prediction of average power and the weekly UCI score were biased as well. Amoukou et al. (2021) presented a solution to this problem by adjusting the highlighted weights in table 10. To account for the correlation for tree 1, one needs to adjust the total observations with respect to $X_2 = 1$ from 1000 to 500, adjust the observations of the subset for $X_1 = 0$ with respect to $X_2 = 1$ from 500 to 100 and adjust the observations of the subset for $X_1 = 1$ with respect to $X_2 = 1$ from 500 to 400. In return, SHAP values for tree methods will be independent from the correlation of two categorical variables and thereby exact as stated by Lundberg and Lee (2017). We recognized the latter findings but were not able to implement them in this work as they have only recently appeared. The author would like to stress that this bias might be limited to those tree ensemble methods which require encoding techniques that extend the feature matrix such as one-hot-encoding. If we understand Amoukou et al. (2021) this bias should not concern results obtained from the Catboost models since Catboost uses target encoding for categorical variables which does not

extend the feature matrix. Furthermore, the symmetric tree-building strategy used by Catboost leads to tree paths that are always symmetric.

Furthermore we would like to stress that a higher stopping criterion than 3 consecutive rounds without RMSE improvement for the Bayesian HPO algorithm could lead to better results for the UCI weekly points prediction. Hvarfner et al. (2022) showed Bayesian HPO can be further improved in terms of time-to-accuracy if one uses a prior belief about a location where an optimum with some hyperparameter settings occurs. The prior belief will be integrated through an adjustment of the acquisition function. Since this prior belief requires some expert's opinion about the dataset and the machine-learning algorithm we suggest, that a possible prior could be generated from literature of comparable machine-learning tasks. If such literature exists one could use the average of each hyperparameter of the given papers as their expert's prior. Another suggestions might be to use the hyperparameter of the most recent paper featuring a comparable machine-learning tasks and the same methods.

7 Conclusion

In this thesis we showed that we can successfully predict the average power of training sessions of professional road cyclists with a precision up to 0.978 for the R^2 metric and an RMSE of 0.1496695 including all variables of the Strava dataset. When we excluded $work\_total$, which was probably calculated using measurements as for the calculation of the average power measurement from the Strava dataset and therefore might have led to biased results, we achieved a R^2 metric in the OOS testing of 0.908 and an RMSE of 0.305. Furthermore we showed that neither a specific season nor a specific cycling type, $sprinter$,$climber$ or $mixed$, helps to improve the average power of a professional road cyclists. Model 3 featured imputed values, using the IRMI algorithm, for the variables of the average temperature and the average calories during the training session. When $work\_total$ was removed from the feature list a drop of almost 8 percentage points of the R^2 value was observed. Future research questions could involve the possibility to replace $work\_total$ with an instrumental variable uncorrelated to the average power measure. This helped the argument that less conservative solutions to handle missing values, e.g. imputing strategies, can be a promising alternative to improve a models prediction quality. We failed to reproduce results with a comparable high performance, predicting the UCI weekly points. The lowest RMSE for the OOS scenario we were able to achieve was 0.982396 which is equal to a R^2 value of 0.982396. SHAP Values helped in both cases to improve the explainability of the mod-

els. For the average power prediction the figures of the SHAP values revealed that the variable $work\_total$ and $move\_time\_sec$ had the biggest impact on the models prediction. For the prediction of the UCI weekly points,figures of the SHAP values of the model 1 fitted with the Lightgbm method and default hyperparameter settings gave us the insight that the variables $seaon$, age, $height$, $avg\_elap\_time\_sec$ showed an overall negative impact. The feature decision plot of the SHAP values indicated that the predictions were much lower than the actual values. This led us to the idea to extract those variables and refit the model with only variables with a possible positive impact on the model. Thereby we achieved an RMSE value of 0.9612489 which equals an R^2 of 0.0685645 an improvement of more than 2 percentage points compared to the models that contained the aforementioned variables with negative impact. The tree SHAP algorithm, as described by Lundberg and Lee (2017), seems to be biased if a non-zero correlation between one-hot-encoded variables exists in a dataset. Following Amoukou et al. (2021) a possible solution to this bias is described but was not implemented. A comparison between the biased and possible unbiased SHAP values of the best models and methods combination the both regression tasks in this thesis could be a possibility for future research. Especially to clarify if the Catboost algorithm with his own target encoding of nominal variable suffers from the same kind of bias as well. Furthermore we have shown that tree-based ensemble methods achieve better prediction results for the average power with the Strava dataset if their hyperparameter were optimized instead of using the default hyperparameter settings. We argue, that using Bayesian HPO for the latter optimization produced on average better results in the same computational time compared to other HPO methods, since a continuous evaluation of the results helped to chose hyperparameter that have a higher chance of coming close to the optimal settings for the regression task. Improvement of this procedure was discussed via the use of an experts belief as a prior. Future research could decrease their effort in terms of computational time to optimize their preferred models.

References

Amoukou, S. I., Brunel, N. J.-B., & Salaün, T. (2021). Accurate shapley values for explaining tree-based models. *AISTATS 2022*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, *13*(null), 281–305.

Brefeld, U., Davis, J., Haaren, J. V., & Zimmermann, A. (2020, December 9). *Machine learning and data mining for sports analytics*. Springer International Publishing. https://www.ebook.de/de/product/41245195/machine_learning_and_data_mining_for_sports_analytics.html

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/a:1010933404324

Brochu, E., Cora, V. M., & de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning.

Chen, T., & Guestrin, C. (2016). Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939785

Claudino, J. G., de Oliveira Capanema, D., de Souza, T. V., Serrão, J. C., Pereira, A. C. M., & Nassis, G. P. (2019). Current approaches to the use of artificial intelligence for injury risk assessment and performance prediction in team sports: A systematic review. *5*(1). https://doi.org/10.1186/s40798-019-0202-3

Hvarfner, C., Stoll, D., Souza, A., Nardi, L., Lindauer, M., & Hutter, F. (2022). Pibo: Augmenting acquisition functions with user beliefs for bayesian optimization. *10th International Conference on Learning Representations, ICLR'22*, 1–30. https://openreview.net/pdf/1ce81b811a1cac6ed2405793a93e8512b1b50005.pdf

Karetnikov, A. (2019). Application of data-driven analytics on sport data from a professional bicycle racing team, 1, 23, 24, 25, 47, 48, 51, 52, 53.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 3149–3157. https://doi.org/10.5555/3294996.3295074

Kholkine, L., Schepper, T. D., Verdonck, T., & Latré, S. (2020). A machine learning approach for road cycling race performance prediction. Springer International Publishing. https://doi.org/10.1007/978-3-030-64912-8_9

Lemaître, G., & Lemaitre, C. (2018). Estimate power without measuring it: A machine learning application (slides). *Science & Cycling 2018 (Conference)*. https://doi.org/10.13140/RG.2.2.20151.57762

Lundberg, S., Erion, G. G., & Lee, S.-I. (2018). Consistent individualized feature attribution for tree ensembles.

Lundberg, S., & Lee, S.-I. (2017). A unified approach to interpreting model predictions.

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). Boosting algorithms as gradient descent., 512–518.

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl.*, *3*(1), 27–32. https://doi.org/10.1145/507533.507538

Mignot, J.-F. (2015). The history of professional road cycling. In *The economics of professional road cycling* (pp. 7–31). Springer International Publishing. https://doi.org/10.1007/978-3-319-22312-4_2

Passfield, L., Hopker, J., Jobson, S., Friel, D., & Zabala, M. (2016). Knowledge is power: Issues of measuring training and performance in cycling. *35*(14), 1426–1434. https://doi.org/10.1080/02640414.2016.1215504

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2019). Catboost: Unbiased boosting with categorical features.

Raghunathan, T. E., Lepkowski, J. M., Hoewyk, J. V., & Solenberger, P. (2001). A multivariate technique for multiply imputing missing values using a sequence of regression models. survey methodology 27.

Ranjan, P., Haynes, R., & Karsten, R. (2011). A computationally stable approach to gaussian process interpolation of deterministic computer simulation data. *Technometrics*, *53*(4), 366–378. https://doi.org/10.1198/tech.2011.09141

Shahhosseini, M., Hu, G., & Pham, H. (2019). Optimizing ensemble weights and hyperparameters of machine learning models for regression problems.

Shapley, L. S. (1952). *A value for n-person games*. RAND Corporation. https://doi.org/10.7249/P0295

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms.

Strnad, D., Nerat, A., & Kohek, Š. (2015). Neural network models for group behavior prediction: A case of soccer match attendance. *28*(2), 287–300. https://doi.org/10.1007/s00521-015-2056-z

Templ, M., Kowarik, A., & Filzmoser, P. (2011). Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, *55*(10), 2793–2806. https://doi.org/10.1016/j.csda.2011.04.012

Tofallis, C. (2015). A better measure of relative prediction accuracy for model selection and model estimation. *66*(8), 1352–1362. https://doi.org/10.1057/jors.2014.103

Wang, X., Zhai, S., & Chen, J.-L. (2020). Research on house price forecast based on hyper parameter optimization gradient boosting regression model. *2020 8th International Conference on Orange Technology (ICOT)*. https://doi.org/10.1109/icot51877.2020.9468726

Wilson, J. T., Hutter, F., & Deisenroth, M. P. (2018). Maximizing acquisition functions for bayesian optimization.

A Appendix

Table 9
Hyperparameter results of the Bayesian HPO of the best model

Method	mtry	Minimal node size	Tree Depth	learning rate	loss reduction
			Model 3 Average Power		
RF	11 ($\lfloor\sqrt{x}\rfloor$*)	-	-	-	
LGBM	15 (all)	15 (20)	5 (no limit)	0.0473691 (0.1)	0.1186154 (0)
XGBoost	**20** (all)	**40** (1)	**14** (6)	**0.0729534** (0.3)	$\mathbf{4.7306132 \times 10^{-4}}$ (0)
Catboost	17 (all)	32 (1)	14 (6)	0.0321339 (0.03)	-
			Model 1 UCI weekly points		
RF	11 ($\lfloor\sqrt{x}\rfloor$*)	2 (5)	-	-	-
LGBM	18 (all)	11 (20)	15 (no limit)	0.0190382 (0.1)	1.3131895×10^{-9} (0)
XGBoost	9 **(all)**	3 **(1)**	14 **(6)**	0.020941 **(0.03)**	5.0319939×10^{-5} **(0)**
Catboost	19 (all)	5 (1)	13 (6)	0.0861325 (0.03)	-

Note. Values in square brackets indicate the default hyperparameter settings. Bold numbers indicate the hyperparameter settings of the method which achieved the lowest RMSE for the best model. '-' indicates that this hyperparameter is not available for tuning. The stop condition of the HPO was 3 consecutive rounds without an RMSE improvement of the model. All methods generated 1000 trees. *x indicates the number of variables of the train dataset.

Table 10
SHAP Value calculation with bias

Tree 1				
Feature subset (S)	$f_x(S)$	$f_x(S \cup \{X_1\})$	$f_x(S \cup \{X_1\}) - f_x(S))$	$\lvert S\rvert!(M-\lvert S\rvert-1)!$
$\emptyset$	100	$\frac{100}{500} * 100 + \frac{400}{500} * 200 = 180$	80	0.5
$\{X_2\}$	$\frac{500}{1000} * 100 + \frac{500}{1000} * 200 = 150$	200	50	0.5
			SHAP value of X_1	90
Feature subset (S)	$f_x(S)$	$f_x(S \cup \{X_2\})$	$f_x(S \cup \{X_2\}) - f_x(S))$	$\lvert S\rvert!(M-\lvert S\rvert-1)!$
$\emptyset$	100	$\frac{500}{1000} * 100 + \frac{500}{1000} * 200 = 150$	50	0.5
$\{X_1\}$	$\frac{100}{500} * 100 + \frac{400}{500} * 200 = 180$	200	20	0.5
			SHAP value of X_2	35
Tree 2				
Feature subset (S)	$f_x(S)$	$f_x(S \cup \{X_1\})$	$f_x(S \cup \{X_1\}) - f_x(S))$	$\lvert S\rvert!(M-\lvert S\rvert-1)!$
$\emptyset$	100	$\frac{500}{1000} * 100 + \frac{500}{1000} * 200 = 150$	50	0.5
$\{X_2\}$	$\frac{100}{500} * 100 + \frac{400}{500} * 200 = 180$	200	20	0.5
			SHAP value of X_1	35
Feature subset (S)	$f_x(S)$	$f_x(S \cup \{X_2\})$	$f_x(S \cup \{X_2\}) - f_x(S))$	$\lvert S\rvert!(M-\lvert S\rvert-1)!$
$\emptyset$	100	$\frac{100}{500} * 100 + \frac{400}{500} * 200 = 180$	80	0.5
$\{X_1\}$	$\frac{500}{1000} * 100 + \frac{500}{1000} * 200 = 150$	200	50	0.5
			SHAP value of X_2	90

Note. SHAP values were calculated using equation 2 of Lundberg and Lee (2017). In case of $|S| = \emptyset$ the author considered $0! = 1$. Highlighted parts of the calculation are weights which cause the bias of the SHAP values, argued by Amoukou et al. (2021).

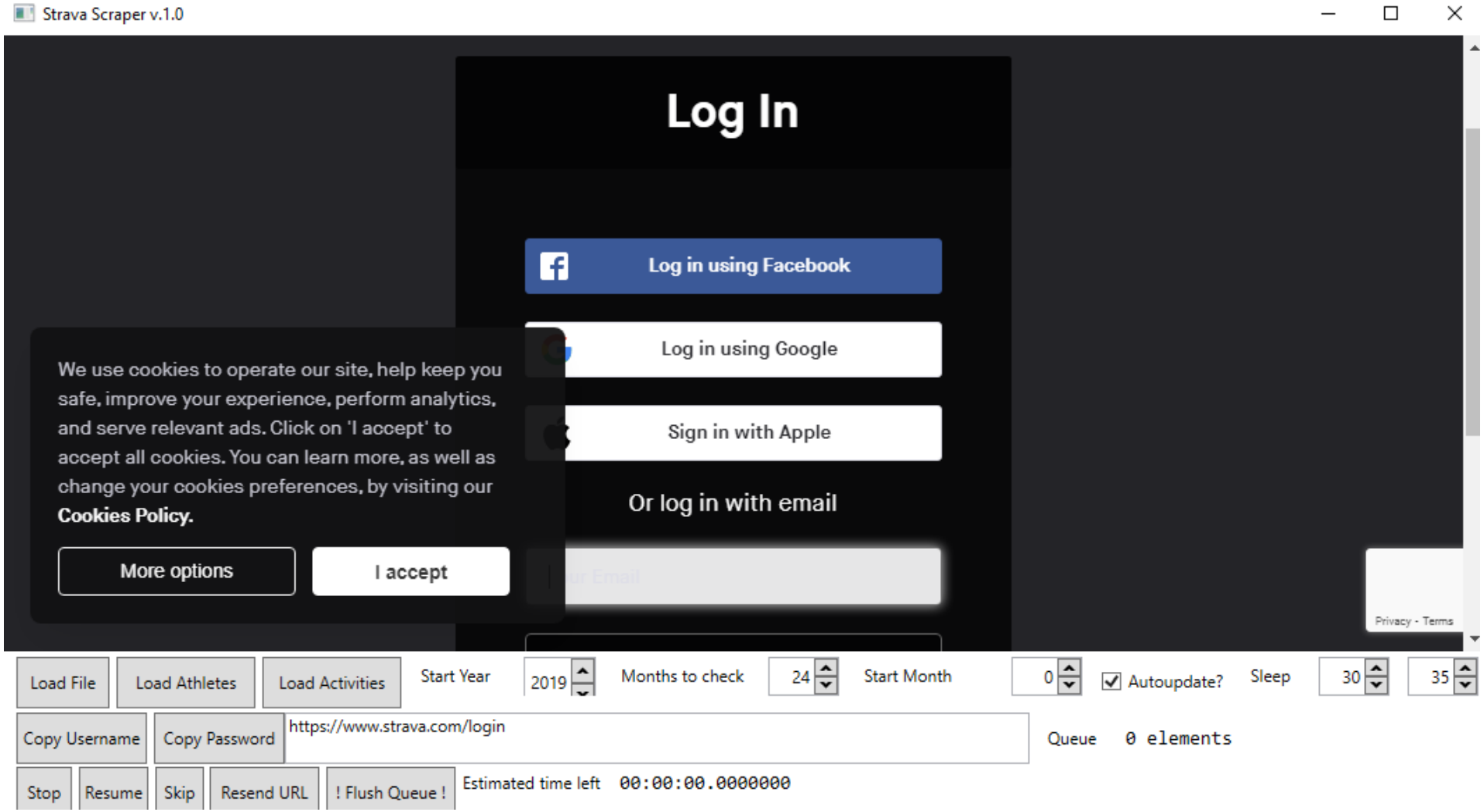


Figure 19: The picture shows the user-interface of the Straver Scraper v.1.0

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Eidesstattliche Versicherung

Hiermit versichere ich an Eides Statt, dass ich diese Arbeit bzw. im Fall einer Gruppenarbeit den von mir entsprechend gekennzeichneten Anteil an der Arbeit selbständig verfasst habe. Ich habe keine unzulässige Hilfe Dritter in Anspruch genommen. Zudem habe ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle Ausführungen (insbesondere Zitate), die anderen Quellen wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht.

Ich versichere, dass die von mir in elektronischer Form eingereichte Version dieser Arbeit mit den eingereichten gedruckten Exemplaren übereinstimmt.

Mir ist bekannt, dass im Falle eines Täuschungsversuches die betreffende Leistung als mit "nicht ausreichend" (5,0) bewertet gilt. Zudem kann ein Täuschungsversuch als Ordnungswidrigkeit mit einer Geldbuße von bis zu 50.000 Euro geahndet werden. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuchs kann ich zudem exmatrikuliert werden.

Mir ist bekannt, dass sich die Prüferin oder der Prüfer bzw. der Prüfungsausschuss zur Feststellung der Täuschung des Einsatzes einer entsprechenden Software oder sonstiger elektronischer Hilfsmittel bedienen kann.

Ich versichere an Eides Statt, dass ich die vorbenannten Angaben nach bestem Wissen und Gewissen gemacht habe und dass die Angaben der Wahrheit entsprechen und ich nichts verschwiegen habe.

Die Strafbarkeit einer falschen eidesstattlichen Versicherung ist mir bekannt, insbesondere die Strafandrohung gemäß §§ 156, 161 StGB, auf welche ich konkret hingewiesen wurde.

§ 156 Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

_____

Ort, Datum Unterschrift