Seminar Summer Semester 2019
Statistical Learning Theory in Pattern Recognition

Statistical Learning Theory Applications: Match Results Prediction of the English Premier League 18-19

Abgabe von Hyovin Kwak
Matrikelnummer 214515
Datenwissenschaft

Contents

1 Introduction **1**
- 1.1 The Premier League . . . 1
- 1.2 Motivation . . . 1
- 1.3 Soccer Prediction Challenge 2017 . . . 1

2 Objectives **1**

3 Inputs and Outputs **1**
- 3.1 Historical Strength . . . 1
- 3.2 Current Form . . . 2
- 3.3 Pi-ratings . . . 2
  - 3.3.1 Definition . . . 2
  - 3.3.2 Measuring error (e) . . . 2
  - 3.3.3 How to assign weight to error (e) . . . 3
  - 3.3.4 How to update Pi-ratings . . . 4
  - 3.3.5 Determining the learning rates and empirical evidence . . . 4
- 3.4 PageRank . . . 5
- 3.5 Match Importance and League . . . 5
- 3.6 Inputs overview . . . 6
- 3.7 Outputs . . . 6

4 Regression and Predictive Models **6**
- 4.1 Overview . . . 6
- 4.2 Regression Models . . . 7
  - 4.2.1 Boosting . . . 7
    - 4.2.1.1 Gradient Tree Boosting . . . 7
    - 4.2.1.2 XGBoost . . . 8
  - 4.2.2 Extremely Randomized Trees(Extra-Trees) . . . 10
  - 4.2.3 Artificial Neural Networks . . . 10
- 4.3 Predictive Models for Classification . . . 11
  - 4.3.1 Beta Distribution . . . 11
  - 4.3.2 Majority Vote Classification: k-Nearest Neighbors . . . 12
  - 4.3.3 Just Classify . . . 13

5 Dataset **13**

6 Results **13**
- 6.1 Default Setting and Tuning Parameters . . . 13
- 6.2 Classification Rate . . . 14

7 Comments and Future Works **14**

8 R Programming **15**

9 Appendix: Data preprocessing **20**

References **31**

1 Introduction

1.1 The Premier League

The Premier League is the top level of the English football league system and one of the most famous football leagues in the world with its wide fan base. 20 clubs compete against each other for a season based on the promotion and relegation system of professional English football league. At the end of every season the change of 3 teams takes place between the Premier League and the championship league, which is the lower division of the Premier League.

1.2 Motivation

Football match results prediction has always been an interesting topic for researchers in the area of machine learning and pattern recognition. It is still an ongoing research topic for many researchers and sports betting companies around the world, in search of the best prediction strategies to predict football match outcomes. One of the recent competitions and its winning solution that we'd like to introduce in this article is the Soccer Prediction Challenge 2017, whose winning solution was presented by Hubáček et al. [1].

1.3 Soccer Prediction Challenge 2017

According to Hubáček et al. [1], the Soccer Prediction Challenge 2017 was organized in conjunction with the MLJ's special issue on Machine Learning for Soccer. The goal of the challenge was to predict match outcomes of future matches within a selected time-period from the different leagues over the world. A dataset of over 200,000 past match outcomes was provided to challenge participants, which includes full-time match scores, date, the name of the home team and its opponent only.

2 Objectives

The purpose of this article consists of two parts: Realization of the winning solution of Hubáček et al. [1] using gradient boosted trees(Xgboost) and comparing the prediction performance of gradient boosted trees to two other machine learning methods. These two methods are Extremely Randomized Trees(Extra-Trees) and Artificial Neural Networks(ANN).

Instead of presenting a prediction model trained by the dataset of over 200,000 match outcomes which is introduced earlier at the section 1.3, the models of this project are trained by 8,740 match outcomes of the Premier Leage seasons from 93-94 until 17-18 in order to predict the match outcomes of the Premier League season 18-19. We try to implement features and methods that are suggested by Hubáček et al. [1] as much as possible to see if we can reach their success. This will be discussed in section 3.

3 Inputs and Outputs

3.1 Historical Strength

Mean and variance of the scored goals, conceded goals, win percentages, and draw percentages are seperately calculated for home and away matches. These statistics are aggregated from the current and last 2 seasons to reflect the historical strength of the teams. When a team is newly promoted from the lower division, the minimum mean values of scored goals and win percentages among the other teams are then assigned to this team, because it is very likely from past experience that a newly promoted team competes for survival in the league, instead of having good chances to win the league title.

3.2 Current Form

Even the strongest teams have a period of ups and downs during a season. Because the features of historical strengths from section 3.1 are rather long-term, Hubáček et al. [1] present other input features which can evaluate the performance of a team from recent match outcomes: current form. A set of statistics is derived from the last 5 matches. This set of statistics, however, will not be computed from home and away matches seperately, because such split might deal with then extremly smaller number of matches. Statistics of current form from the last 5 games include: winning percentage, drawing percentage, goals scored average, goals conceded average, goals scored standard deviation, goals conceded standard deviation and the number of days of rest for the team preceding the match. It is commonly known that the number of days of rest before the match is the factor that tends to affect the performance of a team. The Premier League teams, for example, have to play after taking 1 or 2 days of break during the holiday season in December. Even strong teams may suffer from inconsistent match performance on this period.

3.3 Pi-ratings

3.3.1 Definition

The Pi-rating system developed by Constantinou and Fenton [2] is a rating system that is designed to estimate the expected goal difference of a match on the assumption that both home and away team are playing against a team of an average level. According to Constantinou and Fenton [2], the pi-rating system is built on the following concept. For example, let us assume that the team Y scored 240 goals and conceded 150 goals over the last 100 matches. The team Y has scored then about 90 goals more than it has conceded. If we were to predict Y's goal difference at match instance against an average opponent, the best we could do is to predict +0.9 goals in favor of the team Y and this is the expected goal difference of the team Y against an average opponent. The Pi-rating system updates then this expected value of the goal difference based on three assumptions: the well-known phenomenon of home advantage(Clarke and Norman (1995), Hirotsu and Wright (2003), Poulter (2009)), the fact that most recent results are more important than less recent when estimating current ability (Constantinou et al. (2012b)) and the fact that a win is more important for a team than winning with larger goal differences.

Based on this concept, Constantinou and Fenton [2] define an equation that estimates the expected goal difference for a team on the assumption that it plays against a team of an average level as follows:

$$\widehat{g_{DG}} = b^{\frac{|R_{\tau G}|}{c}} - 1 \tag{1}$$

where $\widehat{g_{DG}}$ describes the expected goal difference for the team against an opponent of an average level that plays at match location G(Home team for H, its opponent for A) and $R_{\tau G}$ denotes the pi-rating of a team τ at match location G. It is to note that b is equal to the logarithm used $b = 10$. It is due to the implementation of the function that assigns weights to error, which will be further discussed in section 3.3.2. Constantinou and Fenton [2] assume here $c = 3$.

3.3.2 Measuring error (e)

Using the equation (1), we are able to calculate the expected goal difference of a match, $\widehat{g_D}$. Although there is no mathematical reason to believe that $\widehat{g_D}$ produces expected goal difference, Constantinou and Fenton [2] accept that the resulting $\widehat{g_D}$ values are useful in earning such a description on the basis of the empirical evidence, which will be further discussed in section 3.3.5., and is measured as follows:

$$\widehat{g_D} = \widehat{g_{DH}} - \widehat{g_{DA}} \tag{2}$$

where a positive value of $\widehat{g_D}$ implies that the home team is expected to win. When it is negative, it implies the opposite. This is considered to be informative and capable of capturing both current form and historical strengths and we can obtain the expected goal difference of a match, subject to the match location. The equation (1) and (2) then lead us to measure the error between the expected goal

difference and the observed goal difference. The observed goal difference g_D is simply the difference of goals that are scored, i.e., for the home team $g_D = g_H - g_A$ where g_H and g_A are the number of goals scored by the home and away team respectively. This error is defined as follows:

$$e = |g_D - \widehat{g_D}| \tag{3}$$

where it is to note that the absolute value of the difference between these two values is considered as the desired error. After the match, pi-ratings are updated for both teams for later use.

3.3.3 How to assign weight to error (e)

According to Constantinou and Fenton [2], one of the important aspects we need to consider when we update the pi-ratings, which will be further discussed in section 3.3.4, is to diminish the effect of large goal differences, such as score results of 5-0 or 6-0 because a win is considered to be more important than a large goal difference. Therefore, Constantinou and Fenton [2] define a function $\psi(e)$ and its derivative is supposed to be decreasing as e increases in a consistent manner. One possible solution, presented by Constantinou and Fenton [2], is a log function of e and defined as:

$$\psi(e) = c \times \log_{10}(1 + e) \tag{4}$$

where they assume that $c = 3$ and e denotes the goal difference error of the equation (3). (See Figure 1)[1][2]

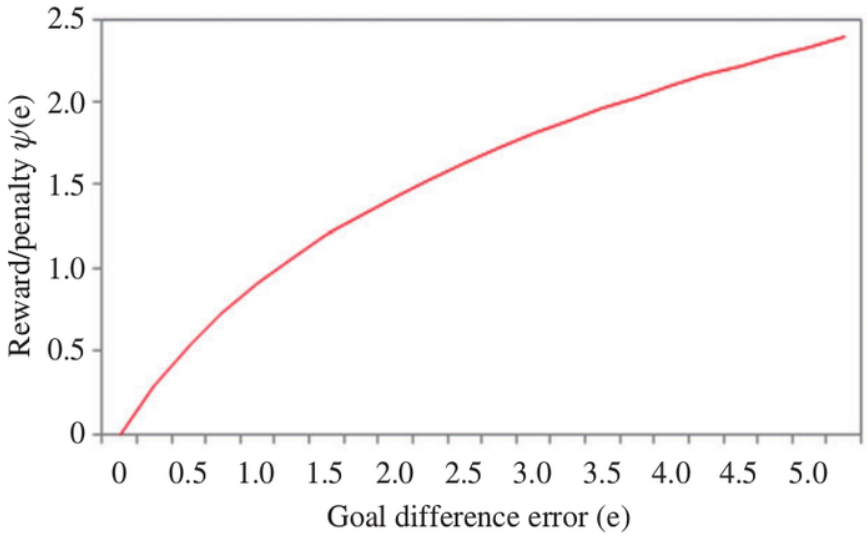

Figure 1: Weighted reward/penalty $\psi(e)$ relative to e , assuming $c = 3$.

The function $\psi(e)$ is further used to define the functions ψ_H and ψ_A, so that the diminishing effect of large goal difference could be finally accomplished:

$$\psi_H(e) = \begin{cases} \psi(e) & : \widehat{g_D} < g_D \\ -\psi(e) & : \text{otherwise} \end{cases}$$

$$\psi_A(e) = \begin{cases} \psi(e) & : \widehat{g_D} > g_D \\ -\psi(e) & : \text{otherwise} \end{cases}$$

[1] According to Constantinou and Fenton[2], we don't know exactly how "less" important each additional score difference may be. This could be one of the future works. They present here just one possible solution. Although it seems random among many choices, the function reflect an important fact that the derivative of this function decreases when the value e increases, in order to make the higher goal differences less influential. Adding 1 to e makes it possible to define the function when $e = 0$.

[2] No more further explanations are given by Constantinou and Fenton [2] to derive $c = 3$. I think this is empirically derived by checking the diminishing effect of high score differences with several choices of c when updating the pi-ratings.

3.3.4 How to update Pi-ratings

The pi-rating can be then updated dynamically in a cumulative way whereby the difference between predicted and observed goal difference determine whether the rating increases or decreases, i.e., a team's rating will increase if the score indicates a higher performance than that predicted by the pi-ratings. Accordingly, the overall rating of a team is the average rating between home and away performances, and this is simply defined as:

$$R_\tau = \frac{R_{\tau H} + R_{\tau A}}{2} \tag{5}$$

where R_τ is the rating for team τ , $R_{\tau H}$ is the rating for team τ at home and $R_{\tau A}$ is the rating of team τ when playing away of its home. Suppose there's a match between home team α and away team β. Then the home and away ratings are updated cumulatively as follows:

$$\widehat{R_{\alpha H}} = R_{\alpha H} + \psi_H(e) \times \lambda \tag{6}$$

$$\widehat{R_{\alpha A}} = R_{\alpha A} + (\widehat{R_{\alpha H}} - R_{\alpha H}) \times \gamma \tag{7}$$

$$\widehat{R_{\beta A}} = R_{\beta A} + \psi_A(e) \times \lambda \tag{8}$$

$$\widehat{R_{\beta H}} = R_{\beta H} + (\widehat{R_{\beta A}} - R_{\beta A}) \times \gamma \tag{9}$$

where $R_{\alpha H}$ and $R_{\alpha A}$ are the current home and away ratings of the team α, $R_{\beta H}$ and $R_{\beta A}$ are the current home and away ratings of the team β. Those $\widehat{R_{\alpha H}}, \widehat{R_{\alpha A}}, \widehat{R_{\beta H}}$ and $\widehat{R_{\beta A}}$ values are newly updated values of the pi-ratings. e is the difference between predicted and observed goal difference. ψ_H and ψ_A are the function of e, which assigns weights to the error e in order to diminish the importance of goal differences. γ is a catch-up learning rate which determines to what extent the newly acquired information based on home performance influences a team's away rating. λ is another learning rate which determines to what extent the newly acquired information of goal-based match results will override the old information in terms of rating. Both γ and λ vary from 0 to 1. How to determine those learning rates will be further discussed in section 3.3.5.

3.3.5 Determining the learning rates and empirical evidence

The learning parameters λ and γ, as mentioned in section 3.3.4, determine to what extent the new information from next matches will override the old information in terms of rating. These parameters can take values from 0 to 1. For instance, when $\lambda = 0.1$ a team's rating will adjust with cumulative updates based on new match results with a weighing factor of 10% and when $\gamma = 0.5$ a team's home performances will affect that team's away ratings with a weighting factor of 50% relative to the revised home rating.

Constantinou and Fenton [2] suggest a way to find optimal learning parameters among possible choices in terms of minimizing the error between the expected and observed goal difference of equation (3). For training the learning parameters they've considered relevant historical data[3]. Accordingly, if a combination of learning rates λ and γ increase the prediction accuracy of e, then we assume that both λ and γ are a step closer to being optimal.

For example, Figure 2 shows how parameters λ and γ affect the squared error in predicted score difference over the Premier League seasons from 1997/1998 to 2006/2007, where the error is simply the difference between predicted and observed goal difference. Minimum squared error is observed when $\lambda = 0.035$ and $\gamma = 0.7$, where $e^2 = 2.6247$.

[3]Football-Data 2012 beginning from season 1992/1993 up to the end of season 2006/2007

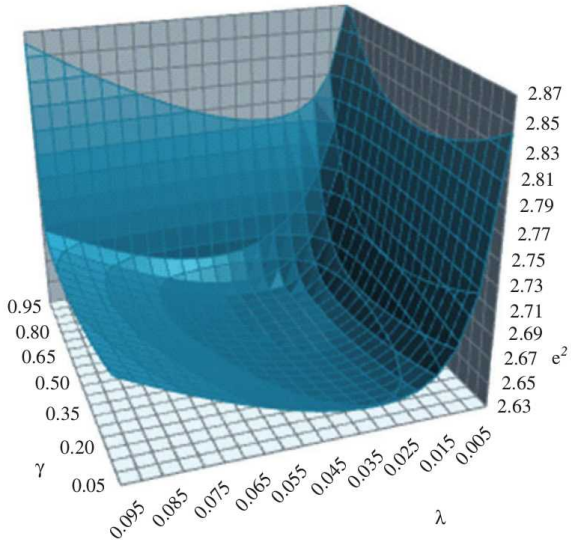

Figure 2: Estimating optimum λ and γ learning rates based on squared goal difference error e^2, over the Premier League seasons from 1997/1998 to 2006/2007.

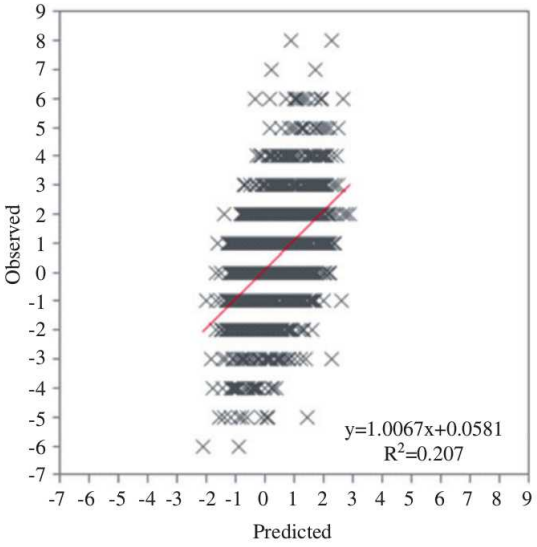

Figure 3: Predicted vs. observed goal difference, over the EPL seasons 2007/2008 to 2011/2012 (1900 match instances)

$\lambda = 0.035$ and $\gamma = 0.7$ provide then the empirical evidence(see Figure 3) that the suggested combination of learning rates provides ratings that accurately capture a team's current performance on the basis of predicted vs. observed goal difference over the next five seasons. The two datasets of Figure 3 seem significantly correlated and therefore the pi-rating system suggested by Constantinou and Fenton [2] seem to be useful indeed, although values in Figure 3 are discrete and there's somehow no guarentee that the observed values around the regression line vary according to normal distribution at the same predicted values. Constantinou and Fenton [2] admit that it is the limitation of pi-rating system to predict the goal differences larger than 3 because of the large variability in observed scores.

3.4 PageRank

A problem of the historical strength from section 3.1 is that it does not take the match results specifically between the two teams in history into account. To solve this problem, another feature called PageRank, suggested by Lazova and Basnarkov [3], is utilized. The PageRank of a team is measured as follows:

$$\frac{3 \times w_{ij} + d_{ij}}{g_{ij}} \tag{10}$$

where $w_{ij}(d_{ij})$ denotes the number of wins(draws) of team i over team j and g_{ij} denotes the number of games played between the two teams. Hubáček et al. [1] utilize match outcomes of previous two seasons and current season to measure the PageRank.

3.5 Match Importance and League

Match importance and league are the features that are also included by Hubáček et al. [1]. Match importance, however, is in fact a combination of long handicraft and subjectivity. Therefore, it is excluded from the inputs. As we are considering only the Premier League, league is also an excluded feature from the inputs.

3.6 Inputs overview

All the input features from section 3.1 to section 3.4 could be summarized in one table as follows:

Historical Strength	**computed from matches from the current and last 2 seasons**
H WIN PCT	Home winning percentage
A WIN PCT	Away winning percentage
H DRAW PCT	Home drawing percentage
A DRAW PCT	Away drawing percentage
H GS AVG	Home goals scored average
A GS AVG	Away goals scored average
H GC AVG	Home goals conceded average
A GC AVG	Away goals conceded average
H GS STD	Home goals scored standard deviation
A GS STD	Away goals scored standard deviation
H GC STD	Home goals conceded standard deviation
A GC STD	Away goals conceded standard deviation
Current Form	**computed from matches from the last 5 matches played**
WIN PCT	Winning percentage
DRAW PCT	Drawing percentage
GS AVG	Goals scored average
GC AVG	Goals conceded average
GS STD	Goals scored standard deviation
GC STD	Goals conceded standard deviation
REST	The number of rest days for the team preceding the match
Pi-ratings	**computed from matches from the current and last 2 seasons**
H RTG	Home pi-rating
A RTG	Away pi-rating
PageRank	**computed from matches from the current and last 2 seasons**
Page Rank	PageRank by Lazova and Basnarkov [3]

Table 1: Summary of Features

which describes total 31 input features of this project.

3.7 Outputs

Full-time results of the matches are the desired output. A win of the home team, draw and a loss of the home team represent three labels of $0, 0.5$ and 1, respectively. This will be further discussed in section 4 to see how this is further modeled after regression so that the outcome of future matches could be predicted.

4 Regression and Predictive Models

4.1 Overview

Hubáček et al. [1] suggest to run the algorithms in a *regression mode* so that the resulting model would yield real numbers $r \in [0, 1]$, as the match outcomes have three categories of $0, 0.5$, and 1. One of the suggestions presented by Hubáček et al. [1] is the XGBoost algorithm by Chen and Guestrin [4], which is one of the algorithmic variations of gradient tree boosting algorithms. Two other methods will also be included in this project: Extremely Randomized Trees by Geurts et al.[5]. and Artificial Neural Networks by Haykin, S (1998) [6] to see if they can show better performance.

After fitting the regression models, according to Hubáček et al. [1], we need to fit predictive models which best fit the labels from the training dataset and predicted values from regression models so that

we finally predict match results. One solution suggested by Hubáček et al. [1] is a predictive model of beta distribution. Not only implementing this predictive model we also decided to implement two more predictive models to see if they show better classification rate: k-Nearest Neighbor and 'Just Classify'. For better understanding, this could be briefly diagrammed as follows:

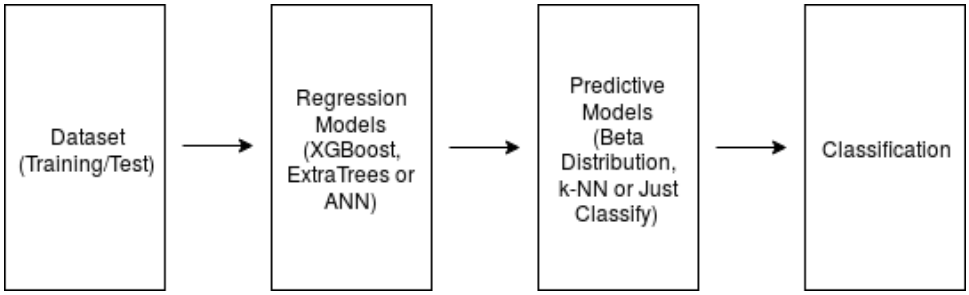

Figure 4: Methods and Predictive Models Flowchart

Note that the classification of the test dataset, which is the last stage of this flowchart, is the objective of this project.

4.2 Regression Models

4.2.1 Boosting

4.2.1.1 Gradient Tree Boosting

For a given dataset size of n and m features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}(|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R})$, a tree ensemble model uses K additive functions to predict the output.

$$\hat{y_i} = f_K(\mathbf{x}_i) = \sum_{k=1}^{K} T_k(\mathbf{x}_i), \quad T_k \in \mathcal{T} \tag{11}$$

where $\mathcal{T} = \{T_k(\mathbf{x})\}$ is the space of (regression) trees. Here we decide to use the notation $\mathcal{T} = \{T_k(\mathbf{x}; \Theta_k)\}$ instead to point out the structure of kth tree with Θ_k, which maps an example to the corresponding leaf index and predict the output.[4]

To build this tree ensemble model of K additive functions, according to Hastie et al. [7], we need to solve an optimization problem to build each tree iteratively

$$\hat{\Theta}_k = \underset{\Theta_k}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y_i, f_{k-1}(x_i) + T_k(x_i; \Theta_k)) \tag{12}$$

where Θ_k denotes the structure of the tree f and $\mathcal{L}$ is a loss function. Here $\hat{\Theta}_k$ is the structure setting of an additional tree, with which the ensemble model minimizes the loss function.

If miminizing of the loss on the training data were the only goal, steepest descent would be the prefered strategy.[5] As our ultimate goal is to generalize $f_K(x)$ to new data, one possible resolution induces a tree $T(\mathbf{x}; \Theta)$ at the kth iteration whose predictions are as close as possible to the negative gradient. Using squared error to measure closeness, this leads us to

$$\tilde{\Theta}_k = \underset{\Theta}{\operatorname{argmin}} \sum_{i=1}^{n} (-g_{ik} - T(x; \Theta))^2 \tag{13}$$

where g_{ik} denotes the gradient of the loss function

[4] Those settings are how Chen and Guestrin [4] start their explanations. Some of variables and minor notations here are changed for convenience in this project.

[5] For more details, please refer to Hastie et al. [7], p. 358

$$g_{ik} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{k-1}(x_i)} \tag{14}$$

It is here to note that the equation (12) with the loss function of squared error is then equivalent to the equation (13), because the additional tree with its structure setting of Θ has the similar direction of the negative gradient, which minimizes the loss function when it is added to the tree ensemble model. The gradient tree boosting algorithm by Hastie et al. [7] is stated below[6].

Algorithm 1 Gradient Tree Boosting Algorithm

Initialize $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1$ **to** M **do**
  for $i = 1$ **to** N **do compute**

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}(x_i)}$$

  end for
  Fit a regression tree to the targets g_{im} giving terminal regions $R_{jm}, j = 1, 2, \cdots J_m$.
  for $j = 1$ **to** J_m **do compute**

$$\gamma_{jm} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

  end for
  Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
end for
Output $\hat{f}(x) = f_M(x)$

4.2.1.2 XGBoost

One of the machine learning methods that are introduced by Hubáček et al. [1] is a scalable tree boosting system, known as XGBoost, by Chen and Guestrin [4]. The scalability of XGBoost is due to several important systems and algorithmic optimizations.[7]

Unlike the gradient tree boosting that we mentioned in section 4.2.1.1, according to Chen and Guestrin [4], we minimize here the following objective

$$\mathcal{L}^{(k)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(k-1)} + f_k(\mathbf{x}_i)) + \Omega(f_k) \tag{15}$$

where $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|w\|^2$. Here l is a differentiable convex loss function that measures the difference between the prediction $\hat{y_i}$ and the target y_i. The second term Ω penalizes the complexity of the model (i.e., the regression tree functions). The additional regularization term helps to smooth the final learnt weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions. This model in equation (15) cannot be optimized using traditional optimization methods in Euclidean space. Instead, the model is trained in an additive manner. Formally, let $\hat{y_i}^{(k)}$ be the prediction of the i-th instance at the kth iteration, we will need to add f_k to minimize the objective.

This means we greedily add the f_k that most improves our model according to equation (15). Second-order approximation can be used to quickly optimize the objective in the general setting.

[6] Hastie et al. [7] p. 361.
[7] For more details, please refer to Chen and Guestrin [4].

$$\mathcal{L}^{(k)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y_i}^{(k-1)} + g_i f_k(\mathbf{x}_i)) + \frac{1}{2}h_i f_k^2(\mathbf{x}_i)] + \Omega(f_k) \tag{16}$$

where $g_i = \frac{\partial}{\partial z} l(y_i, z)\Big|_{z=\hat{y_i}^{(k-1)}}$ and $h_i = \frac{\partial^2}{\partial y^2} l(y_i, z)\Big|_{z=\hat{y_i}^{(k-1)}}$ are the first- and second-order gradients (partial derivatives) of the local objective evaluated at $\hat{y_i}^{(k-1)}$. After removing the constant terms, Chen and Guestrin [4] show that we obtain the following simplified objective at step k.

$$\tilde{\mathcal{L}}^{(k)} = \sum_{i=1}^{n}[g_i f_k(\mathbf{x}_i)) + \frac{1}{2}h_i f_k^2(\mathbf{x}_i)] + \Omega(f_k) \tag{17}$$

For a fixed structure Θ, this equation (17) ultimately leads to the equation of

$$\tilde{\mathcal{L}}_{\Theta}^{(t)} = -\frac{1}{2}\sum_{j=1}^{V}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma V \tag{18}$$

which can be used as a scoring function for the Algorithm 2 to measure the quality of a tree structure Θ. Here V denotes the number of leaves in the tree. This score is like the impurity score, such as misclassification error, the Gini index or Cross-entropy[8], for evaluating decision trees, except that it is derived for a wider range of objective functions.

A greedy algorithm which starts from a single leaf and iteratively adds branches to the tree is used instead. Assume that I_L and I_R are the instance sets of left and right nodes after the split. Letting $I = I_L \cup I_R$, then the loss reduction after the split is given by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{19}$$

which indicates in practice best split in Algorithm 2 among split candidates.

One of the split finding algorithms we'd like to introduce is basic exact greedy algorithm, which enumerates all the possible splits and returns the split with max score.

Algorithm 2 Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
  $gain \leftarrow 30$
  $G \leftarrow \sum_{i\in I} g_i$, $H \leftarrow \sum_{i\in I} h_i$
  for $k = 1$ **to** n **do**
    $G_L \leftarrow 0, H_L \leftarrow 0$
    for j *in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
      $G_L \leftarrow G_L + g_i$, $H_L \leftarrow H_L + h_j$
      $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
      $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end for
  end for
Output: Split with max score

Usually it is impossible to enumerate all the possible splits. This leads to other algorithmic innovations of XGBoost by Chen and Guestrin [4] such as the approximate algorithm to effectively find best splitting points among candidates.

[8]For specific definitions, please refer to Hastie et al. [7], p. 308-309.

4.2.2 Extremely Randomized Trees(Extra-Trees)

The other tree-based ensemble method we'd like to introduce for this project is Extremely Randomized Trees by Geurts et al. [5].

Algorithm 3 Extra-Trees splitting algorithm (for numerical attributes)

Split_a_node(S) ▷ *the local learning subset S corresponding to the node we want to split*
- If **Stop_split**(S) is TRUE then return nothing.
- Otherwise select K attributes $\{a_1, \cdots, a_K\}$ among all non constant (in S) candidate attributes;
- Draw K splits $\{s_1, \cdots, s_K\}$, where $s_i =$ **Pick_a_random_split**$(S, a_i), \forall i = 1, \cdots, K$;
- return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\cdots,K} \text{Score}(s_i, S)$.

▷ *returns a split $[a < a_c]$ or nothing*

Pick_a_random_split(S, a)[9] ▷ *a subset S and an attribute a*
- Let a^S_{max} and a^S_{min} denote the maximal and minimal value of a in S.
- Draw a random cut-point a_c uniformly in $[a^S_{min}, a^S_{max}]$
- Return the split $[a < a_c]$

Stop_split(S) ▷ *a subset S*
- If $|S| < n_{min}$, then return TRUE;
- If all attributes are constant in S, then return TRUE;
- If the output is constant in S, then return TRUE;
- Otherwise, return FALSE.

where the $\text{Score}(s_*, S)$ function in this project is defined as:

$$\text{Score}(s_*, S) = \frac{var\{y|S\} - \frac{|S_l|}{|S|} var\{y|S_l\} - \frac{|S_r|}{|S|} var\{y|S_r\}}{var\{y|S\}} \tag{20}$$

$var\{y|S\}$ denotes the variance of the output y in the sample S. A random attribute and a random cut-point at each node by **Pick_a_random_split** build totally randomized trees whose structures are independent of the target variable values of the learning sample. Note that random cut-points are uniformly distributed. For more detailed information, please see the presentation by Teschke [12].

4.2.3 Artificial Neural Networks

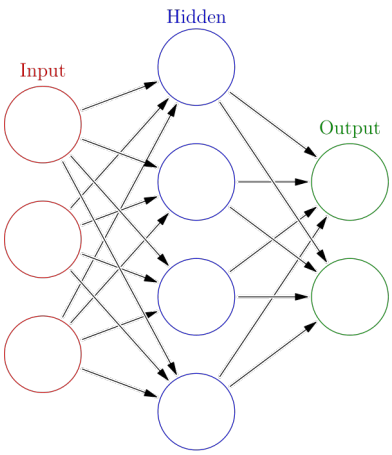

Figure 5: Artificial Neural Networks

Another method we'd like to compare with those two tree-based methods is the traditional artificial neural networks. Here we also run the algorithms in *regression mode* to see if its prediction performance

[9] attribute in this context means feature.

is better than those two tree-based methods. For more detailed information, such as feed-forward or back propagation, please see the presentation by Rosenbohm and Westhoff [13].

4.3 Predictive Models for Classification

4.3.1 Beta Distribution

Hubáček et al. [1] suggest to run the algorithms that are mentioned in section 4.2 in a *regression mode* so that the resulting model would yield real numbers $r \in [0, 1]$, as the match outcomes have three categories of $0, 0.5$ and 1. Then they introduce one more predictive model for classification. According to Hubáček et al. [1], for each label $i \in \{w, d, l\}$ it is assumed that[10]

$$P_i(r) = \frac{f_i(r)}{f_w(r) + f_d(r) + f_l(r)} \tag{21}$$

where r for each label i is modeled to have beta distribution[11].

$$f_i(r) = \frac{r^{\alpha-1}(1-r)^{\beta-1}}{B(\alpha, \beta)} \tag{22}$$

where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$, $\alpha > 0$, $\beta > 0$ and Γ is the Gamma function, which is according to the Wackerly et al [14], $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}dx$. The parameters α and β should maximize the function's fit with tuples $(r, P_i(r))$ available in training data. In particular, $P_i(r)$ is the proportion of training examples with outcome i among all examples for which the model yields r.[12] At the end of the process, the label of the predicted values r_m will be estimated based on the following rule:[13]

$$\widehat{Label(r_m)} = \underset{i \in \{w,d,l\}}{\operatorname{argmax}} P_i(r_m) \tag{23}$$

In order to implement the predictive model of the equation (21), we use method of moments estimation according to Owen [9] and Buchholz [10] in which the parameters of a distribution are displayed as a function of the moments. Let us assume that we have a sample $(r_1, ...r_n)$ size of n where every value r_m is the realization of a random variable R_m and all random variables R_m are independent and identically distributed. We define then $\tilde{R}^k$ as an estimator for $E(R^k)$ and $\hat{R}^k$ as a concrete estimated value. The estimator defined as below is unbiased:

$$\tilde{R}^k = \frac{1}{n}\sum_{j=1}^{n}(R_j)^k \tag{24}$$

and its estimated value can be defined as:

$$\hat{R}^k = \frac{1}{n}\sum_{j=1}^{n}(r_j)^k \tag{25}$$

An unbiased estimator $\tilde{S}^2$ and the estimated value $\hat{S}^2$ for the variance are defined respectively as:

$$\tilde{S}^2 = \frac{1}{n-1}\sum_{j=1}^{n}(R_j - \tilde{R}^1)^2 \tag{26}$$

[10] w stand for win, d stands for draw and l stands for loss

[11] No references by Hubáček et al. [1] are given. According to Albert and Koning[11], however, such modeling is common and possible because beta distribution is conjugate prior to binomial distribution. Similar example of such modeling could be found in batting average of baseball players. As the batting average is one of the season-long statistics, it is often estimated to be beta distribution and updated after the player's chance at batter's box, which is again then beta-distributed. They mention for further reading Morrison and Kalwani(1993, Chapter 7)

[12] Intuitively, e.g. $f_w(r)$ is an estimate of the win-probability for a match with regressor's output r. Note that in general $f_l(r) + f_d(r) + f_w(r) \neq 1$, hence the $f_i(r)$ is normalized in equation (21)

[13] The notation $Label(r_m)$ here means the label of r_m for convenience.

$$\hat{S}^2 = \frac{1}{n-1}\sum_{j=1}^{n}(r_j - \hat{R}^1)^2 \tag{27}$$

We also know that the mean and variance of a beta distributed random variable R can be defined as:

$$E[R] = \frac{\alpha}{\alpha+\beta} \tag{28}$$

$$Var[R] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{29}$$

So the parameters of the beta distribution in our context can be then estimated as the function of the moments. We use the estimated values of estimated values for $\tilde{R}^1$ and $\tilde{S}^2$ as:

$$\hat{R}^1 = \frac{\alpha}{\alpha+\beta} \tag{30}$$

$$\hat{S}^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{31}$$

In order to obtain α and β, we first solve for β of the equation (30). After solving for β we obtain

$$\beta = \frac{\alpha}{\hat{R}^1} - \alpha \tag{32}$$

Then we plug this β in the equation (31). To do so, we first reshape the equation (31) which becomes

$$\alpha\beta = \hat{S}^2(\alpha+\beta)^2(\alpha+\beta+1) \tag{33}$$

Plugging the equation (32) into in the equation (33) and then solving for α will be then

$$\alpha = \hat{R}^1\left(\frac{\hat{R}^1(1-\hat{R}^1)}{\hat{S}^2} - 1\right) \tag{34}$$

Finally, we again plug this in the equation (32) to obtain beta as the function of the moments of estimated values.

$$\beta = (1-\hat{R}^1)\left(\frac{\hat{R}^1(1-\hat{R}^1)}{\hat{S}^2} - 1\right) \tag{35}$$

Our method of moments estimated values for α and β are therefore:

$$\hat{\alpha}_{MOM} = \hat{R}^1\left(\frac{\hat{R}^1(1-\hat{R}^1)}{\hat{S}^2} - 1\right) \tag{36}$$

$$\hat{\beta}_{MOM} = (1-\hat{R}^1)\left(\frac{\hat{R}^1(1-\hat{R}^1)}{\hat{S}^2} - 1\right) \tag{37}$$

4.3.2 Majority Vote Classification: k-Nearest Neighbors

According to Hubáček et al. [1], the idea of the beta dstribution modelling is that its parameters α and β maximize the function's fit with tuples r, $P_i(r)$ available in training data. In particular, $P_i(r)$ is the proportion of training examples with outcome i among all examples for which the model yields r. Keeping this in mind, we'd like to present a non-parametric solution that maps the results of regression to the desired label $i \in \{w, d, l\}$, the k-nearest neighbors algorithm, which estimates labels based on the distance to the nearest neighbors.

4.3.3 Just Classify

We also would like to see how with no certain predictive model but just mapping the predicted values to labels using the function

$$f(r) = \begin{cases} 1, & \frac{2}{3} \leq r < 1 \\ 0.5, & \frac{1}{3} \leq r < \frac{2}{3} \\ 0, & 0 \leq r < \frac{1}{3} \end{cases}$$

could reach classification rate better than two other methods. Here 1 denotes a win, 0.5 a draw and 0 a loss.

5 Dataset

Training dataset we'd like to use in this project is a set of match results of the Premier League season from 93-94 to 17-18, which is a set of 8,740 games. Test dataset is a set of match results of the season 18-19, which consists of 380 games. For each dataset we use team names and full-time match only to derive features that were mentioned in section 3.

6 Results

6.1 Default Setting and Tuning Parameters

We train the model with the training dataset using those three methods as mentioned in section 4. First we run the XGBoost algorithm by Chen and Guestrin [4] which is suggested by Hubáček et al. [1] in *regression mode* to build a desired tree ensemble model. After the implementation of XGBoost regression, we try to see if we find the best combination among sets of important tuning parameters, max_depth(maximum depth of a tree) and eta(learning rate). Default setting of XGBoost in R is first implemented, its number of round($nrounds$) being 300 with $eta = 0.5$ and $max\_depth = 2$. To find the best combination of tuning parameters the 5-fold-cross-validation method were repeated 100 times by randomly choosing the parameters from $max\_depth \in \{6, 7, 8, 9, 10\}$, $eta \in [0.01, 0.3]$ and $nrounds \in \{100, 200, 300, 400, 500, 600, 700, 800, 900\}$.

Next, we run the Extra-Trees algorithm to build another tree ensemble model. Unfortunately, it was not successful to execute cross-validation to find the parameters of best performance due to memory error.[14]. We tried several sets of possible parameters of $nrounds \in \{100, 200, 300, 400, 500, 600, 700\}$ and $nodesize \in \{2, 3, 4, 5, 6, 7\}$.

Models of artificial neural networks is the last model we wanted to compare with. Finding a suitable hidden layer is an important factor when implementing neural networks. Here we randomly select hidden layer candidates, considering only one or two layers. The number of hidden neurons of each layer is also randomly selected from $\{1, 2, 3, 4, 5\}$.

Default setting of XGBoost, which is stated in Table 2, with $max\_depth = 2$, $eta = 0.5$ and $nrounds = 300$ resulted in the mean squared error of 0.1895 between the labels of 0, 0.5 and 1.0 from test dataset and the predicted values. The Extra-Trees algorithm, on the other hand, showed the mean squared error of 0.1588 with its default setting of $nrounds = 500$ and $nodesize = 5$. The method of the Best.Neural Networks showed the best performance of its mean squared error. With only one hidden layer with 2 neurons in it showed the mean squared error of 0.1509. This method, however, is not cross-validated.

[14]The implementation with multiple parameter settings unfortunately leads to memory error, I think it is due to memory error of R using Java.

Table 2: Mean Squared Errors

Methods	MSE	Parameters
Default.XGBoost	0.1895	$max\_depth = 2, eta = 0.5, nrounds = 300$
Best.XGBoost	0.3941	$max\_depth = 6, eta = 0.012, nrounds = 300$
Default.Extra-Trees	0.1588	$nrounds = 500, nodesize = 5, seed\_number = 3927$
Best.Extra-Trees	0.1620	$nrounds = 100, nodesize = 5, seed\_number = 7659$
Default.Neural Networks	0.1519	$hidden = c(1)$
Best.Neural Networks	0.1509	$hidden = c(2)$

6.2 Classification Rate

Based on the outcomes with different parameter settings from section 6.1, we classify the predicted values using different methods that are mentioned in section section 4.3. Classification rate is defined as follows:

$$Classification\,rate = \frac{1}{n}\sum_{j=1}^{n} I(\widehat{Label(r_j)} = Label(r_j)) \tag{38}$$

Classification rates of the methods that are mentioned in section 6.1 are shown in Table 3. $k = 20$ was selected for the k-nearest-neighbor algorhtm. Although predictive model assuming beta distribution looked quite acceptable, the non-parametric solution, the k-nearest-neighbor algorithm, shows better results for about $5 - 10\%$ when it comes to using the XGBoost algorithm. Neural Networks showed good classification rates in general in this project.

Table 3: Classification Rates

Methods	Beta Distribution	k-Nearest-Neighbors	Just Classify
Default.XGBoost	0.434	0.539	0.407
Best.XGBoost	0.502	0.560	0.439
Default.Extra-Trees	0.428	0.397	0.444
Best.Extra-Trees	0.413	0.394	0.410
Default.Neural Networks	0.555	0.544	0.481
Best.Neural Networks	0.557	0.555	0.471

Although Extra-Trees are considered to be robust (Geurts et al. [5]) with its *nodesize* = 5 in *regression mode*, the classification rate of it during this project is not successful, which seems over-fitted. As we can see in Figure 6, the predicted values from training dataset by regression are good organized. Predicted values of test dataset by regression, however, seems over-fitted, because its minimum value is around 0.22 among all 380 match outcomes.

7 Comments and Future Works

Over the course of this project, several ideas are to mention as well. Firstly, for the method of moments that is mentioned in section 4.3.1 it is assumed that the sample is independent and identically distributed. On this assumption we estimated those parameters. This could've been better if we could somehow test our sample to see if independent and identically distributed. Especially, we may test the sample if it is independent because the there's possibility of being correlated for the given sample. For example, if one key player of the team is injured during a match and not available for the next match, this could definitely affect the performance of the team. This kind of situation, however, is not modeled as a random variable in this project. This could also lead to the additional discussion to see if other methods and models would be more effective. As the matches are held over the course of time, it could be an option to analyze them using recurrent neural networks or LSTM which could model time-series data. According to Buchholz [10], this could be also followed up by

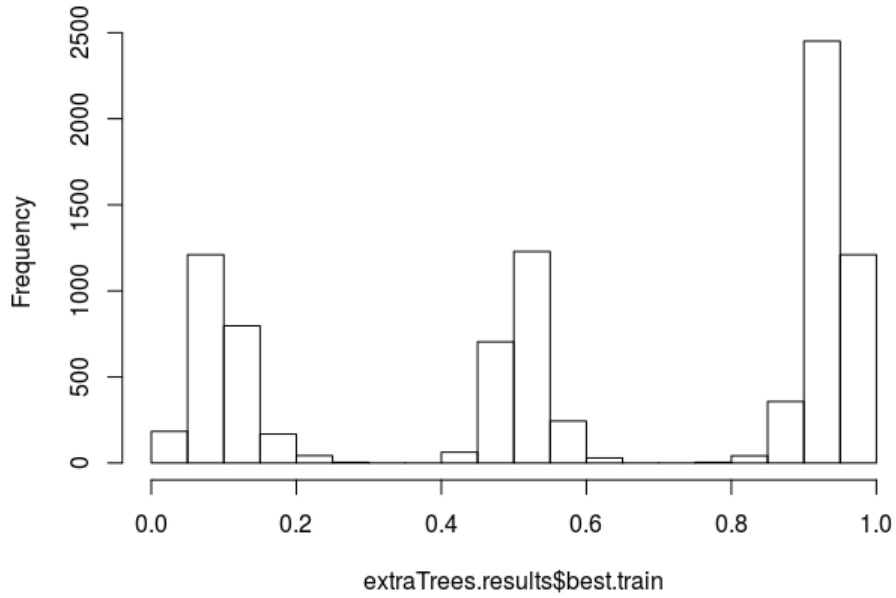

Figure 6: A Histogram of Best.Extra-Trees

other tests, for example, Smirnov-Kolmogorov test, which is a nonparametric test that can be used to compare a sample with a reference probability distribution, such as beta distribution, to test if the values r for each label i are beta distributed.

Secondly, the project of Hubáček et al. [1] could've been better if they had explicitly described how they estimated the parameters mentioned in section 4.3.1. They only mention that the regression results r could be modeled with beta distribution, without mentioning the estimation methods such as method of maximum likelihood or method of moments, which is mentioned in section 4.3.1.

8 R Programming

```
#############################################################
# 0. (optional) Load libraries and working directory check #
#############################################################
source("Appendix.R")
library(remotes)    # These three are used by tr.dataset() and ts.dataset()
library(piratings)  # from Appendix.
library(lubridate)  #

library(xgboost)    # XGBoost
library(rJava)      # extraTrees
library(extraTrees) # extraTrees
library(neuralnet)  # NeuralNetworks
library(svMisc)     # for loop process check
library(class)      # k-nearest-neighbors

#################
# 1. Load data #
#################
# CAUTION: Before you run the code below, please make sure that you already
#          saved the files at a same directory.
#          The matrices are already built by tr.dataset() and ts.dataset()
```

```
#         from Appendix1.R
load("tr_mat.RData") # tr.dataset()
load("ts_mat.RData") # ts.dataset()

#########################
# 2. XGBoost-Regression #
#########################
# 2.1 Default Settings
dtrain <- xgb.DMatrix(data = tr.data[,1:31], label = tr.data[,32])
dtest <- xgb.DMatrix(data = ts.data[,1:31], label = ts.data[,32])
xgboost.results <- list("default.rmse" = 0,
                        "default.pred" = c(),
                        "default.train" = 0,
                        "best.rmse" = Inf,
                        "best.train" = 0,
                        "best.rmse.index" = 0,
                        "best.seednumber" = 0,
                        "best.nround" = 0,
                        "best.pred" = c(),
                        "best.max_depth" = 0,
                        "best.eta" = 0)
param <- list(max_depth = 2,
               eta = 0.5,
               objective = "reg:squarederror",
               eval.metric = "error")
watchlist <- list(train=dtrain, test=dtest)
bst <- xgb.train(param, dtrain, nrounds = 300, watchlist = watchlist)
xgboost.results$default.train <- bst
xgboost.results$default.rmse <- mean((ts.data[,32] - predict(bst, dtest))^2)
xgboost.results$default.pred <- predict(bst, dtest)

# 2.2 Parameter Tuning example: max_depth, eta(learning rate)
for(i in 1:50) {
  # 100 times of Cross-Validation, randomly selected parameters.
  seed.number <- sample.int(10000, 1)
  set.seed(seed.number)
  progress(i)
  Sys.sleep(0.01)
  if(i == 100){cat("Done!\n")}
  param <- list(booster = "gbtree",
                objective = "reg:squarederror",
                eval_metric = "rmse",
                max_depth = sample(6:10, 1),
                eta = runif(1, .01, .3),
                subsample = 1)
  cv.nround <- sample(c(100, 200, 300, 400, 500, 600, 700, 800, 900), 1)
  cv.nfold <- 5
  mdcv <- xgb.cv(data=dtrain, params = param, nfold=cv.nfold,
                 nrounds=cv.nround, verbose = 0)
  min.error <- min(mdcv[4]$evaluation_log$test_rmse_mean)
  min.error.index <- which.min(mdcv[4]$evaluation_log$test_rmse_mean)

  if(min.error < xgboost.results$best.rmse){
    xgboost.results$best.rmse <- min.error
    xgboost.results$best.rmse.index <- min.error.index
    xgboost.results$best.seednumber <- seed.number
    xgboost.results$best.nround <- mdcv$niter
    xgboost.results$best.max_depth <- mdcv$params$max_depth
    xgboost.results$best.eta <- mdcv$params$eta
  }
}
```

```
# compute best prediction
set.seed(xgboost.results$best.seednumber)
param <- list(max_depth = xgboost.results$best.max_depth,
              eta = xgboost.results$best.eta,
              objective = "reg:squarederror",
              eval.metric = "error")
best.train <- xgb.train(param, dtrain, nrounds = xgboost.results$best.nround, watchlist)
xgboost.results$best.pred <- predict(best.train, dtest)
xgboost.results$best.train <- best.train

# trim the prediction values. (0, 1)
xgboost.results$default.pred[xgboost.results$default.pred > 1] <- 0.999999
xgboost.results$default.pred[xgboost.results$default.pred < 0] <- 0.000001
xgboost.results$best.pred[xgboost.results$best.pred > 1] <- 0.999999
xgboost.results$best.pred[xgboost.results$best.pred < 0] <- 0.000001

save(xgboost.results, file = "xgboost_results.RData") ## save data.


############################
# 3. extraTrees-Regression #
############################
options(java.parameters = "-Xmx4g") # set the java memory

# 3.1 Default Setting:
seed.number <- sample.int(10000, 1)
set.seed(seed.number)
extraTrees.results <- list("default.rmse" = 0,
                           "default.pred" = c(),
                           "default.seednumber" = 0,
                           "default.train" = 0,
                           "best.rmse" = Inf,
                           "best.nodesize" = 0,
                           "best.seednumber" = 0,
                           "best.ntree" = 0,
                           "best.pred" = 0,
                           "best.train" = 0)
extraTrain <- extraTrees(tr.data[,1:31], tr.data[,32], ntree = 500, nodesize = 5)
extraTrees.results$default.pred <- predict(extraTrain, as.matrix(ts.data[,1:31]))
extraTrees.results$default.rmse <- mean((extraTrees.results$default.pred - ts.data[,32])^2)
extraTrees.results$default.seednumber <- seed.number
extraTrees.results$default.train <- predict(extraTrain, as.matrix(tr.data[,1:31]))

# 3.2 Parameter Tuning: try different best.ntree and best.nodesize.
# CAUTION: extraTrees doesn't work with for-loops! No cross-validation.
seed.number <- sample.int(10000, 1)
set.seed(seed.number)
best.ntree <- 100
best.nodesize <- 5
extraTrain <- extraTrees(tr.data[,1:31], tr.data[,32], ntree = 100, nodesize = 5)
extraTrees.temp.pred <- predict(extraTrain, as.matrix(ts.data[,1:31]))
extraTrees.temp.rmse <- mean((extraTrees.temp.pred - ts.data[,32])^2)
if(extraTrees.temp.rmse < extraTrees.results$best.rmse){
  extraTrees.results$best.rmse <- extraTrees.temp.rmse
  extraTrees.results$best.nodesize <- best.nodesize
  extraTrees.results$best.seednumber <- seed.number
  extraTrees.results$best.pred <- extraTrees.temp.pred
  extraTrees.results$best.ntree <- best.ntree
  extraTrees.results$best.train <- predict(extraTrain, as.matrix(tr.data[,1:31]))
}
# trim the values.
```

```
extraTrees.results$default.pred[extraTrees.results$default.pred > 1] <- 0.999999
extraTrees.results$default.pred[extraTrees.results$default.pred < 0] <- 0.000001
extraTrees.results$best.pred[extraTrees.results$best.pred > 1] <- 0.999999
extraTrees.results$best.pred[extraTrees.results$best.pred < 0] <- 0.000001

save(extraTrees.results, file = "extraTrees_results.RData") ## save data.


#################################
# 4. Neural Networks-Regression #
#################################
nn.results <- list("default.rmse" = 0,
                   "default.pred" = c(),
                   "default.train" = 0,
                   "best.rmse" = Inf,
                   "best.pred" = c(),
                   "best.hidden" = c(),
                   "best.seednumber" = 0,
                   "best.train" = 0)
# 4.1 Scale Data
maxs <- apply(tr.data[,1:32], 2, max)
mins <- apply(tr.data[,1:32], 2, min)
tr.scaled <- as.data.frame(scale(tr.data[,1:32], center = mins, scale = maxs - mins))
ts.scaled <- as.data.frame(scale(ts.data[,1:32], center = mins, scale = maxs - mins))
names(tr.scaled)[30:31] <- c("PageRank_H", "PageRank_A")
names(ts.scaled)[30:31] <- c("PageRank_H", "PageRank_A")

# 4.2 generate Model
n <- names(tr.scaled)
f <- as.formula(paste("FTR ~", paste(n[!n %in% "FTR"], collapse = " + ")))

# 4.3 Neural Networks regression: threshold = 0.15, stepmax = 1.1e+05
nn.train <- neuralnet(f, data=tr.scaled,stepmax = 1.1e+05,
                      linear.output=TRUE, threshold = 0.15, lifesign = "full")
nn.results$default.train <- nn.train
nn.results$default.pred <- as.vector(compute(nn.train, as.matrix(ts.scaled[,1:31]))$net.result)
nn.results$default.rmse <- mean((nn.results$default.pred - ts.data[,32])^2)

# 4.4 Parameter Tuning: hidden layers
for(i in 1:100){ ## iterate 100 times
  seed.number <- sample.int(10000, 1)
  set.seed(seed.number)
  progress(i)
  Sys.sleep(0.01)
  if(i == 100){cat("Done!\n")}

  layers <- sample(1:2, 1)
  hidden_i <- sample(1:5, layers)
  nn.train <- neuralnet(f, data=tr.scaled,stepmax = 1.1e+05, hidden = hidden_i,
                        linear.output=TRUE, threshold = 0.15, lifesign = "full")
  pred <- as.vector(compute(nn.train, as.matrix(ts.scaled[,1:31]))$net.result)

  if(mean((pred - ts.data[,32])^2) < nn.results$best.rmse){
    nn.results$best.rmse <- mean((pred - ts.data[,32])^2)
    nn.results$best.hidden <- hidden_i
    nn.results$best.seednumber <- seed.number
    nn.results$best.pred <- pred
    nn.results$best.train <- nn.train
  }
}
nn.results$default.pred[nn.results$default.pred > 1] <- 0.999999
```

```
nn.results$default.pred[nn.results$default.pred < 0] <- 0.000001
nn.results$best.pred[nn.results$best.pred > 1] <- 0.999999
nn.results$best.pred[nn.results$best.pred < 0] <- 0.000001
save(nn.results, file = "nn_results.RData")


###############
# 5. Results #
###############
# 5.1 Load data
load("xgboost_results.RData")
load("extraTrees_results.RData")
load("nn_results.RData")

# a list of predictions from test data.
results.list <- list(xgboost.results$default.pred, ## don't forget. these are
                     xgboost.results$best.pred, ## test data!
                     extraTrees.results$default.pred,
                     extraTrees.results$best.pred,
                     nn.results$default.pred,
                     nn.results$best.pred)
# a list of trainig. Please note that they are the training results of
# xgboost and neural networks. We need one more function predict(data, model)
# to calculate actual values.
# extraTrees are, however, actual results of training.
train.list <- list(xgboost.results$default.train,
                   xgboost.results$best.train,
                   extraTrees.results$default.train, # <- these two are vectors
                   extraTrees.results$best.train, # of prediction(tr.data)
                   nn.results$default.train,
                   nn.results$best.train)
results.names <- c("Default.XGBoost",
                   "Best.XGBoost",
                   "Default.extraTrees",
                   "Best.extraTrees",
                   "Default.NN",
                   "Best.NN")
# 5.1 Beta Distribution Modelling: Hubacek's idea, use tr.data.
for(i in 1:6){
  pred <- 0
  ## take the prediction values from calculation between model and tr.data
  if(i == 3 || i == 4 ){ ## extraTrees. They are already calculated.
    pred <- train.list[[i]]
  }else if(i == 5 || i == 6 ){ ## neural network.
    pred <- as.vector(compute(train.list[[i]], as.matrix(tr.scaled[,1:31]))$net.result)
  }else{ ## XGBoost
    pred <- predict(train.list[[i]], tr.data[,1:31])
  }
  ## make sure that the values are in (0, 1)
  pred[pred > 1] <- 0.999999
  pred[pred < 0] <- 0.000001

  ## Use mean and variance to estimate thetas of beta distribution
  ## (method of moments)
  thetas <- a.b.rechner(c(mean(pred[tr.data[, 32] == 0.0]),
                          var(pred[tr.data[, 32] == 0.0]),
                          mean(pred[tr.data[, 32] == 0.5]),
                          var(pred[tr.data[, 32] == 0.5]),
                          mean(pred[tr.data[, 32] == 1.0]),
                          var(pred[tr.data[, 32] == 1.0])))
  results.list[[i]][results.list[[i]] > 1] <- 0.999999
```

```
  results.list[[i]][results.list[[i]] < 0] <- 0.000001
  print.this <- paste(results.names[i],
                      "classification rate:",
                      mean(WDL.prediction(results.list[[i]], thetas) == ts.data[,32]))
  print(print.this)
}

# 5.2 Majority Vote - K-Nearest Neighbors
for(k in 20){
  for(i in 1:6){
    r.vec <- results.list[[i]]
    pred <- 0
    ## take the prediction values from calculation between model and tr.data
    if(i == 3 || i == 4 ){
      pred <- train.list[[i]]
    }else if(i == 5 || i == 6 ){
      pred <- as.vector(compute(train.list[[i]], as.matrix(tr.scaled[,1:31]))$net.result)
    }else{
      pred <- predict(train.list[[i]], tr.data[,1:31])
    }
    ## make sure that the values are in (0, 1)
    pred[pred > 1] <- 0.999999
    pred[pred < 0] <- 0.000001
    KNN <- knn(train = as.matrix(pred),
               test = as.matrix(results.list[[3]]),
               cl = as.factor(tr.data[,32]), k)
    print.this <- paste(results.names[i],
                        "classification rate:",
                        mean(KNN == ts.data[,32]), "// k:", k)
    print(print.this)
  }
}

# 5.3 No model, just classify: (0, 0.333), (0.333, 0.666), (0.666, 0.999)
for(i in 1:6){
  r.vec <- results.list[[i]]
  memory.vec <- rep(0, length(r.vec))
  memory.vec[which(0 < r.vec & r.vec < 0.333333)] <- 0
  memory.vec[which(0.333333 < r.vec & r.vec < 0.666666)] <- 0.5
  memory.vec[which(0.666666 < r.vec & r.vec < 1)] <- 1
  print.this <- paste(results.names[i],
                      "classification rate:",
                      mean(memory.vec == ts.data[,32]))
  print(print.this)
}

# 5.4 Histogram, extraTrees
hist(extraTrees.results$best.train, main = "")
hist(results.list[[4]], main = "")
min(results.list[[4]])
```

9 Appendix: Data preprocessing

```
#### Overview:
# 1. Load training and test dataset.
#    tr.dataset()
#    ts.dataset()
#    tr.outcomes()
```

```
# 2. functions to build tr.dataset, ts.dataset.
#    (load data, utility functions, features.. etc.)
# 3. functions for results classification



#### Functions:
# tr.dataset   - builds a training dataset.
#
# Inputs :
#   wd         - working directory of file 9318.csv
#                (ex: "/home/Thomas/Documents")
# Outputs:
#  memory.mat  - a data frame of training dataset.
tr.dataset <- function(wd){
  M <- read.9318(wd)
  start <- match.num.rechner(1995)[1]
  end <- match.num.rechner(2017)[2]
  memory.mat <- matrix(0, nrow = (end - start + 1), ncol = 32)
  k <- 1
  for(i in start:end){
    print(i)
    home <- M[i,]$HomeTeam
    away <- M[i,]$AwayTeam
    histStr <- hStr(i, M)
    memory.mat[k, 1:12] <- histStr # 12
    memory.mat[k, 13:19] <- current.form(i, team = home, M) # 7
    memory.mat[k, 20:26] <- current.form(i, team = away, M) # 7
    memory.mat[k, 27:29] <- pi_rating(i, M) # home pi, away pi, expectedGD # 3
    memory.mat[k, 30:31] <- HtH(i, M[i,]$HomeTeam, M[i,]$AwayTeam, M) # 2
    print(M[i, ]$FTR)
    memory.mat[k, 32] <- M[i, ]$FTR
    k <- k + 1
  }
  colnames(memory.mat) <- c( # HistoricalStrength Home/Away
    "H.W.PCT","A.W.PCT","H.D.PCT","A.D.PCT",
    "H.GS.AVG","A.GS.AVG","H.GC.AVG","A.GC.AVG",
    "H.GS.STD","A.GS.STD","H.GC.STD","A.GC.STD",
    #  CurrentForm-Home
    "H.W.PCT5","H.D.PCT5","H.GS.AVG5","H.GC.AVG5",
    "H.GS.STD5","H.GC.STD5","H.REST",
    #  CurrentForm-Away
    "A.W.PCT5","A.D.PCT5","A.GS.AVG5","A.GC.AVG5",
    "A.GS.STD5","A.GC.STD5","A.REST",
    # Pi-ratings
    "H.RTG","A.RTG","EGD",
    # PageRank
    "PageRank H","PageRank A",
    "FTR")
  memory.mat[memory.mat[,"FTR"] == 1,][,"FTR"] <- 0 ## L
  memory.mat[memory.mat[,"FTR"] == 2,][,"FTR"] <- 0.5 ## D
  memory.mat[memory.mat[,"FTR"] == 3,][,"FTR"] <- 1 ## W
  return(memory.mat)
}

# ts.dataset   - builds a test dataset.
#
# Inputs :
#   wd         - working directory of file 1819.csv
#                (ex: "/home/Thomas/Documents")
# Outputs:
```

```
#  memory.mat  - a data frame of test dataset.
ts.dataset <- function(wd){

  tsM <- read.1819(wd) ## read test_dataset:
  tsM <- merge(tsM[,-(11:62)], "2018-19")
  M <- read.9318(wd)

  colnames(tsM) <- colnames(M)
  all.M <- rbind(M, tsM)

  start <- match.num.rechner(2018)[1]
  end <- match.num.rechner(2018)[2]
  memory.mat <- matrix(0, nrow = (end - start + 1), ncol = 32)
  k <- 1
  for(i in start:end){
    print(i)
    home <- all.M[i,]$HomeTeam
    away <- all.M[i,]$AwayTeam
    memory.mat[k, 1:12] <- hStr(i, all.M)
    memory.mat[k, 13:19] <- current.form(i, team = home, all.M) # 7
    memory.mat[k, 20:26] <- current.form(i, team = away, all.M) # 7
    memory.mat[k, 27:29] <- pi_rating(i, all.M) # home pi, away pi, expectedGD # 3
    memory.mat[k, 30:31] <- HtH(i, all.M[i,]$HomeTeam, all.M[i,]$AwayTeam, all.M) # 2
    print(all.M[i, ]$FTR)
    memory.mat[k, 32] <- all.M[i, ]$FTR
    k <- k + 1
  }
  colnames(memory.mat) <- c( # HistoricalStrength Home/Away
    "H.W.PCT","A.W.PCT","H.D.PCT","A.D.PCT",
    "H.GS.AVG","A.GS.AVG","H.GC.AVG","A.GC.AVG",
    "H.GS.STD","A.GS.STD","H.GC.STD","A.GC.STD",
    #  CurrentForm-Home
    "H.W.PCT5","H.D.PCT5","H.GS.AVG5","H.GC.AVG5",
    "H.GS.STD5","H.GC.STD5","H.REST",
    #  CurrentForm-Away
    "A.W.PCT5","A.D.PCT5","A.GS.AVG5","A.GC.AVG5",
    "A.GS.STD5","A.GC.STD5","A.REST",
    # Pi-ratings
    "H.RTG","A.RTG","EGD",
    # PageRank
    "PageRank H","PageRank A",
    "FTR")
  memory.mat[memory.mat[,"FTR"] == 1,][,"FTR"] <- 0 ## L
  memory.mat[memory.mat[,"FTR"] == 2,][,"FTR"] <- 0.5 ## D
  memory.mat[memory.mat[,"FTR"] == 3,][,"FTR"] <- 1 ## W
  return(memory.mat)
}

# tr.outcomes  - builds labels of training dataset.
#
# Inputs :
#   wd         - working directory of file 9318.csv
#                (ex: "/home/Thomas/Documents")
# Outputs:
#             - a data frame of labels of training dataset.
tr.outcomes <- function(wd){
  M <- read.9318(wd)
  return(M[match.num.rechner(1995)[1]:match.num.rechner(2017)[2],
           c("HomeTeam", "AwayTeam", "FTR")])
}
```

```
# read.1819   - loads a data frame of season data 18-19
# Inputs :
#  wd         - working directory of where the file 1819.csv is
#               (ex: "/home/Thomas/Documents")
# Outputs:
#             - data frame of season data 18-19
read.1819 <- function(wd){
  setwd(wd)
  filename <- "1819.csv"
  EPLSeasons <- list(1)
  EPL1819 <- read.csv(filename)
  EPL1819$HomeTeam <- as.character(EPL1819$HomeTeam)
  EPL1819$AwayTeam <- as.character(EPL1819$AwayTeam)
  return(EPL1819)
}

# read.9318   - loads a data frame of season data from 93-94 to 17-18
# Inputs :
#  wd         - working directory of where the file 1819.csv is
#               (ex: "/home/Thomas/Documents")
# Outputs:
#             - data frame of season data from 93-94 to 17-18
read.9318 <- function(wd){ ## it was once load.data()
  setwd(wd)
  M <- read.csv("9318.csv")
  M$HomeTeam <- as.character(M$HomeTeam)
  M$AwayTeam <- as.character(M$AwayTeam)
  # there is typo. fix Middelsboro.
  M[M$HomeTeam == "Middlesboro", ]$HomeTeam <- "Middlesbrough"
  M[M$AwayTeam == "Middlesboro", ]$AwayTeam <- "Middlesbrough"
  return(M)
}

# restdays    - calculates the gap between two match dates.
# Inputs :
#  matchday1  - combination of string and integer, the first match date.
#  matchday2  - combination of string and integer, the next match date.
# Outputs:
#             - integer, the gab between two match dates: "days of rest"
restdays <- function(matchday1, matchday2){
  library(lubridate)
  matchday1 <- as.Date(parse_date_time(matchday1, c('dmy')))
  matchday2 <- as.Date(parse_date_time(matchday2, c('dmy')))
  return(abs(as.numeric(matchday2 - matchday1)))
}

# fromRear    - returns TRUE indexes from rear.
# Inputs :
#  TF         - a boolean vector
#
# Outputs:
#             - the indexes of last 5 "TRUE"s
fromRear <- function(TF){
  A <- c()
  N <- length(TF)
  count <- 0
  for(i in N:1){
    if(TF[i]){ ## if TRUE
      count <- count + 1
      A <- c(i, A)
    }
```

```
    if(count == 5){
      return(A)
    }
  }
  return(A)
}

# name_generator - returns a vector of team names.
# Inputs :
#  teams         - names of teams with duplicates from all matches.
#
# Outputs:
#               - a vector of sorted team names, without duplicates.
name_generator <- function(teams){
  names <- c()
  for(i in teams){
    if(sum(names == i) == 0){
      names <- c(names, i)
    }
  }
  return(sort(names))
}

# match.num.rechner - returns start and end index of the k-(k+1) season.
#          (if k = 1993, match index from 93-94 season will be from 1 to 462)
# Inputs :
#  k               - a positive integer in [1993, 2017]
#
# Outputs:
#                  - a vector of returns start and end index the season
match.num.rechner <- function(k){
  games.num <- c(462, 462, rep(380, 23), 380)
  k <- k - 1993 + 1
  if(k == 1){
    start <- 1
    end <- 462
  }else{
    start <- 1 + sum(games.num[1:(k-1)])
    end <- sum(games.num[1:(k)])
  }
  return(c(start, end))
}

# season.rechner - calculates the season number to which the match
#                  number belongs.
#
# Inputs :
#  match.num     - a positive integer of match number.
#
# Outputs:
#    k           - a season number which corresponds to the match number.
season.rechner <- function(match.num){
  for(k in 1993:2018){
    if(match.num.rechner(k)[1]<= match.num &&
       match.num <= match.num.rechner(k)[2]){
      return(k)
    }
  }
}

# hStr        - calculates historical strength features last 2 seasons.
```

```
#
# Inputs :
#  match.num   - match number of target match.
#      M       - Match dataset
#
# Outputs:
#  memory.vec  - a vector of historical strength features.
hStr <- function(match.num, M){
  ## : returns historical WDLs of all teams during previous 2 two seasons.
  this.season <- season.rechner(match.num)
  start <- match.num.rechner(this.season - 2)[1]
  end <- match.num-1

  target.M <- M[start:end,]
  M <- M[start:(end+1), ]
  teams <- name_generator(c(M$HomeTeam, M$AwayTeam))
  teams.list <- list()

  WDL.mat <- matrix(0, nrow = length(teams), ncol = 12)
  rownames(WDL.mat) <- teams

  i.collector <- c()
  for(i in 1:length(teams)){ ## for each teams,
    team <- teams[i]
    if(sum(target.M$HomeTeam == team) >= 2 && sum(target.M$AwayTeam == team) >= 2){

      HW <- sum(target.M$HomeTeam == team & target.M$FTR == "H")
      HD <- sum(target.M$HomeTeam == team & target.M$FTR == "D")
      HG <- sum(target.M$HomeTeam == team)
      HGOALS <- target.M[target.M$HomeTeam == team, ]$FTHG
      HGOALC <- target.M[target.M$HomeTeam == team, ]$FTAG

      AW <- sum(target.M$AwayTeam == team & target.M$FTR == "A")
      AD <- sum(target.M$AwayTeam == team & target.M$FTR == "D")
      AG <- sum(target.M$AwayTeam == team)
      AGOALS <- target.M[target.M$AwayTeam == team, ]$FTAG
      AGOALC <- target.M[target.M$AwayTeam == team, ]$FTHG

      WDL.mat[i, 1] <- (HW/HG)*100
      WDL.mat[i, 2] <- (AW/AG)*100
      WDL.mat[i, 3] <- (HD/HG)*100
      WDL.mat[i, 4] <- (AD/AG)*100
      WDL.mat[i, 5] <- mean(HGOALS)
      WDL.mat[i, 6] <- mean(AGOALS)
      WDL.mat[i, 7] <- mean(HGOALC)
      WDL.mat[i, 8] <- mean(AGOALC)
      WDL.mat[i, 9] <- sqrt(var(HGOALS))
      WDL.mat[i, 10] <- sqrt(var(AGOALS))
      WDL.mat[i, 11] <- sqrt(var(HGOALC))
      WDL.mat[i, 12] <- sqrt(var(AGOALC))#

      i.collector <- c(i.collector, i)
    }
  }
  for(j in (1:length(teams))[-i.collector]){
    WDL.mat[j,1] <- min(WDL.mat[i.collector, 1])
    WDL.mat[j,2] <- min(WDL.mat[i.collector, 2])
    WDL.mat[j,3] <- mean(WDL.mat[i.collector, 3])
    WDL.mat[j,4] <- mean(WDL.mat[i.collector, 4])

    WDL.mat[j,5] <- min(WDL.mat[i.collector, 5])
```

```
    WDL.mat[j,6] <- min(WDL.mat[i.collector, 6])
    WDL.mat[j,7] <- max(WDL.mat[i.collector, 7])
    WDL.mat[j,8] <- max(WDL.mat[i.collector, 8])


    WDL.mat[j,9] <- max(WDL.mat[i.collector, 9])
    WDL.mat[j,10] <- max(WDL.mat[i.collector, 10])
    WDL.mat[j,11] <- max(WDL.mat[i.collector, 11])
    WDL.mat[j,12] <- max(WDL.mat[i.collector, 12])
  }

  Home <- M[nrow(M),]$HomeTeam
  Away <- M[nrow(M),]$AwayTeam
  memory.vec <- rep(0, 12)

  for(i in 1:12){
    if(i%%2 == 1){
      memory.vec[i] <- WDL.mat[Home,][seq(1, 12, 2)][(i+1)/2]
    }else{
      memory.vec[i] <- WDL.mat[Away,][seq(2, 12, 2)][i/2]
    }
  }
  return(memory.vec)
}

# pi_rating    - calculates pi-ratings from 2 previous seasons.
#
# Inputs :
#  match.num   - match number of target match.
#      M       - Match dataset
#
# Outputs:
#      a       - pi-rating of home team
#      b       - pi-rating of away team
#      c       - expected goal difference between the two teams.
pi_rating <- function(match.num, M){
  library(piratings)

  this.season <- season.rechner(match.num)
  start <- match.num.rechner(this.season - 2)[1]
  end <- match.num-1

  outcomes <- as.matrix(M[start:end, c("FTHG", "FTAG")])
  teams <- as.matrix(M[start:end, c("HomeTeam", "AwayTeam")])

  lambda <- 0.06
  gamma <- 0.5

  piratings <- calculate_pi_ratings(teams, outcomes, lambda, gamma)

  teams <- as.matrix(M[start:end+1, c("HomeTeam", "AwayTeam")])
  team.names <- name_generator(c(teams[,1], teams[,2]))

  result.mat <- matrix(0, nrow = length(team.names), ncol = 2)
  rownames(result.mat) <- team.names
  colnames(result.mat) <- c("pi_Home", "pi_Away")

  for(i in 1:length(team.names)){

    if(sum(teams[,1] == team.names[i]) == 0 && sum(teams[,2] == team.names[i]) == 0){
      result.mat[i,1] <- 0
```

```
      result.mat[i,2] <- 0
    }else if(sum(teams[,1] == team.names[i]) > 0 && sum(teams[,2] == team.names[i]) == 0){
      home.pi.index <- max(which(teams[,1] == team.names[i]))
      result.mat[i,1] <- piratings[home.pi.index, 1]
      result.mat[i,2] <- 0
    }else if(sum(teams[,1] == team.names[i]) == 0 && sum(teams[,2] == team.names[i]) > 0){
      away.pi.index <- max(which(teams[,2] == team.names[i]))
      result.mat[i,1] <- 0
      result.mat[i,2] <- piratings[away.pi.index, 2]
    }else{
      home.pi.index <- max(which(teams[,1] == team.names[i]))
      away.pi.index <- max(which(teams[,2] == team.names[i]))
      result.mat[i,1] <- piratings[home.pi.index, 1]
      result.mat[i,2] <- piratings[away.pi.index, 2]
    }
  }
  HomeTeam <- M[match.num,]$HomeTeam
  AwayTeam <- M[match.num,]$AwayTeam

  a <- result.mat[which(rownames(result.mat) == HomeTeam), 1]
  b <- result.mat[which(rownames(result.mat) == AwayTeam), 2]
  c <- 10^(a/3) - 1 - (10^(b/3) - 1) ## Expected GD.
  return(c(a, b, c))
}

# current.form - calculates current form of the team, from last 5 games.
#
# Inputs :
#  match.num   - match number of target match.
#     team     - the name of the team
#      M       - Match dataset
#
# Outputs:
#             - a vector of current form features.
current.form <- function(match.num, team, M){
  start <- match.num-81
  end <- match.num-1
  target <- M[start:end,]$HomeTeam == team | M[start:end,]$AwayTeam == team

  if(sum(target) == 0){
    return(c(0, 0, 0, 0, 0, 0, 7))
  }else if(sum(target) == 1){

    last.5games <- M[start:end,][fromRear(target),]
    current.5 <- c()
    for(i in 1:nrow(last.5games)){
      if(last.5games[i,]$HomeTeam == team){
        if(last.5games[i,]$FTR == "H"){
          current.5 <- c("W", current.5)
        }else if(last.5games[i,]$FTR == "D"){
          current.5 <- c("D", current.5)
        }else{
          current.5 <- c("L", current.5)
        }
      }else{ ## last.5games[i,]$AwayTeam == team
        if(last.5games[i,]$FTR == "H"){
          current.5 <- c("L", current.5)
        }else if(last.5games[i,]$FTR == "D"){
          current.5 <- c("D", current.5)
        }else{
          current.5 <- c("W", current.5)
```

```
      }
    }
  }
  if(nrow(last.5games[last.5games$HomeTeam == team,]) == 1){ #Heimspiel
    last.5games.GF <- c(last.5games[last.5games$HomeTeam == team,]$FTHG)
    last.5games.GA <- c(last.5games[last.5games$HomeTeam == team,]$FTAG)
  }else{ #Aussp.
    last.5games.GF <- c(last.5games[last.5games$AwayTeam == team,]$FTAG)
    last.5games.GA <- c(last.5games[last.5games$AwayTeam == team,]$FTHG)
  }
  return(c(mean(current.5 == "W") * 100,
           mean(current.5 == "D") * 100,
           last.5games.GF,
           last.5games.GA,
           0,
           0,
           restdays(M[match.num,]$Date, last.5games[nrow(last.5games), ]$Date))
  )
 }else{
  last.5games <- M[start:end,][fromRear(target),]
  current.5 <- c()
  for(i in 1:nrow(last.5games)){
    if(last.5games[i,]$HomeTeam == team){
      if(last.5games[i,]$FTR == "H"){
        current.5 <- c("W", current.5)
      }else if(last.5games[i,]$FTR == "D"){
        current.5 <- c("D", current.5)
      }else{
        current.5 <- c("L", current.5)
      }
    }else{ ## last.5games[i,]$AwayTeam == team
      if(last.5games[i,]$FTR == "H"){
        current.5 <- c("L", current.5)
      }else if(last.5games[i,]$FTR == "D"){
        current.5 <- c("D", current.5)
      }else{
        current.5 <- c("W", current.5)
      }
    }
  }
  last.5games.GF <- c(last.5games[last.5games$HomeTeam == team,]$FTHG,
                      last.5games[last.5games$AwayTeam == team,]$FTAG)
  last.5games.GA <- c(last.5games[last.5games$HomeTeam == team,]$FTAG,
                      last.5games[last.5games$AwayTeam == team,]$FTHG)
  return(c(mean(current.5 == "W") * 100,
           mean(current.5 == "D") * 100,
           mean(last.5games.GF),
           mean(last.5games.GA),
           sqrt(var(last.5games.GF)),
           sqrt(var(last.5games.GA)),
           restdays(M[match.num,]$Date, last.5games[nrow(last.5games), ]$Date))
  )
 }
}

# HtH         - calculates head-to-head statistic(PageRank)
#
# Inputs :
#  match.num   - match number of target match.
#    Home      - name of the home team
#    Away      - name of the away team
```

```
#     M       - Match dataset
#
# Outputs:
#             - PageRank statistic
HtH <- function(match.num, Home, Away, M){

  this.season <- season.rechner(match.num)

  start <- match.num.rechner(this.season - 2)[1]
  end <- match.num-1

  teams <- name_generator(c(M[start:end+1,]$HomeTeam, M[start:end+1,]$AwayTeam))
  HTH.mat <- matrix(0, nrow = length(teams), ncol = length(teams))

  colnames(HTH.mat) <- teams
  rownames(HTH.mat) <- teams

  M <- M[start:end,]
  head2head.comb <- combn(1:length(teams), 2)
  for(i in 1:ncol(head2head.comb)){

    team1 <- teams[head2head.comb[,i][1]]
    team2 <- teams[head2head.comb[,i][2]]

    HG <- sum(M$HomeTeam == team1 & M$AwayTeam == team2)
    HW <- sum(M$HomeTeam == team1 & M$AwayTeam == team2 & M$FTR == "H")
    HD <- sum(M$HomeTeam == team1 & M$AwayTeam == team2 & M$FTR == "D")
    HL <- sum(M$HomeTeam == team1 & M$AwayTeam == team2 & M$FTR == "A")

    AG <- sum(M$AwayTeam == team1 & M$HomeTeam == team2 )
    AW <- sum(M$AwayTeam == team1 & M$HomeTeam == team2 & M$FTR == "A")
    AD <- sum(M$AwayTeam == team1 & M$HomeTeam == team2 & M$FTR == "D")
    AL <- sum(M$AwayTeam == team1 & M$HomeTeam == team2 & M$FTR == "H")

    if(HG+AG >= 1){
      HTH.mat[team1, team2] <- (3*HW + HD + 3*AW + AD) / (HG + AG)
      HTH.mat[team2, team1] <- (3*HL + HD + 3*AL + AD) / (HG + AG)
    }else{
      HTH.mat[team1, team2] <- 0
      HTH.mat[team2, team1] <- 0
    }
  }
  return(c(HTH.mat[Home, Away], HTH.mat[Away, Home]))
}

### functions for results classification:

### functions for results classification:
# a.b.rechner  - estimate parameters of beta distribution, using
#                method of moments.
#
# Inputs :
#   mean.var    - vector of mean and variance of each labels
#                 c(W_mean, W_var, D_mean, D_var, L_mean, L_var)
# Outputs:
#               - a vector of estimation of alpha and betas, which are the
#                 parameters of beta distribution for each label(W, D, L).
a.b.rechner <- function(mean.var){
  memory.vec <- rep(0, length(mean.var))
  for(j in (seq(1, length(mean.var)/2)*2 - 1)){ ## take the mean values first.
```

```
    # method of moments alpha and beta estimation
    a <- mean.var[j]*( (mean.var[j]*(1-mean.var[j]))/mean.var[j+1] - 1)
    b <- (1-mean.var[j])*( (mean.var[j]*(1-mean.var[j]))/mean.var[j+1] - 1)

    memory.vec[j] <- a
    memory.vec[j+1] <- b
  }
  return(memory.vec)
}

# WDL.prediction - predicts the outcomes
#
# Inputs :
#   pred     - prediction value from regression, real number in (0, 1)
#  thetas    - a length 6 vector of estimated alpha and betas from a.b.rechner
#              index 1, 3 and 5 are alphas for W, D, L respectively,
#              index 2, 4 and 6 are betas for W, D, L respectively.
# Outputs:
#           - a vector of predictions, according to the Prediction Rule.
WDL.prediction <- function(pred, thetas){

  memory.vec <- rep(0, length(pred))

  for(i in 1:length(pred)){

    fl <- dbeta(pred[i], thetas[1], thetas[2])
    fd <- dbeta(pred[i], thetas[3], thetas[4])
    fw <- dbeta(pred[i], thetas[5], thetas[6])

    pl <- fl / (fw + fd + fl)
    pd <- fd / (fw + fd + fl)
    pw <- fw / (fw + fd + fl)

    k <- which.max(c(pw, pd, pl))
    ## Prediction Rule
    if(k == 1){ ## win!
      memory.vec[i] <- 1
    }else if(k == 2){
      memory.vec[i] <- 0.5
    }else{
      memory.vec[i] <- 0
    }
  }
  return(memory.vec)
}
```

References

[1] Hubáček, O., Šourek, G. and Železný, F. (2019) *Learning to predict soccer results from relational data with gradient boosted trees.* Mach Learn 108:29–47.

[2] Constantinou, A. C. and Fenton, N. E. (2013) *Determining the level of ability of football teams by dynamic ratings based on the relative discrepancies in scores between adversaries.* Journal of Quantitative Analysis in Sports, 9(1), 37–50.

[3] Lazova, V. and Basnarkov, L. (2015). *PageRank approach to ranking national football teams.* arXiv preprint, arXiv:1503.01331.

[4] Chen, T. and Guestrin, C. (2016) *XGBoost: A Scalable Tree Boosting System.* KDD'16, 785-794.

[5] Geurts, P., Damien, E. and Wehenkel, L.(2006) *Extremely randomized trees.* Mach Learn 63: 3–42

[6] Haykin, S. (1998) *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall

[7] Hastie, T. et al. (2009) *The Elements of Statistical Learning*, 2nd ed. Springer

[8] Datasets available at:
https://www.kaggle.com/thefc17/epl-results-19932018
https://datahub.io/sports-data/english-premier-league

[9] Owen, C. (2008) *Parameter Estimation for the Beta Distribution*, Master Thesis, Brigham Young University

[10] Buchholz, P. (2019) *Modellgestützte Analyse und Optimierung*, Lecture Script from the lecture "Modellgestützte Analyse und Optimierung", Technische Universität Dortmund

[11] Albert, J. and Koning, R. (2019) *Statistical Thinking in Sports*, 1st Edition, Chapman & Hall/CRC

[12] Teschke, S. (2019) *Extremely Randomized Trees, Eine Erweiterung von Random Forests*, Presentation, Seminar Summer Semester 2019 Statistical Learning Theory in Pattern Recognition, TU Dortmund, Germany

[13] Rosenbohm, N. and Westhoff, J. *Neuronale Netze in der Bilderverarbeitung*, Presentation, Seminar Summer Semester 2019 Statistical Learning Theory in Pattern Recognition, TU Dortmund, Germany

[14] Wackerly, D., Mendenhall, W. and Scheaffer, R. *Mathematical Statistics with Applications*, 7th Edition, Thomson Brooks/Cole