CAS ADS Final Project

Predicting the number of scored goals of a football game

June 2023

Nicolas Gfeller

nicolas.gfeller@insel.ch

Abstract

For several years, I have been following football leagues all around Europe. However, since a while, I have not been interested in the outcome of a game (winner / loser) anymore but rather in the analytics behind it. Is the outcome of a football game predictable? And if so, in what sense?

In this project, those questions are elaborated and evaluated by the help of applied machine learning methods. More specifically, supervised machine learning regression models are applied as prediction models. Those models are described in theory in one part and then applied to a real case in another one. Based on those models, the number of goals of a future football game are predicted.

There has been a lot of research done on projects about predicting the outcome of a football game, like predicting a win, lose, or draw between two teams [1], [2], [3]. Thus, as such, there is not a distinctive difference between this project and already available knowledge. However, in this project the outcome of a football game is not asked for. The focus is only on predicting the number of goals per game or the probability of a certain outcome.

Table of Contents

1. Introduction

The project presented in this paper aims at predicting the number of goals scored in a random football game. The required time series for this regression problem is sourced from an online data provider, and machine learning as well as deep learning methods are applied in order to find the best fit. For this paper, the data was taken from the Swiss Super League. However, as the data format is steady and clean, the model can be applied to any other league's dataset which is available.

In the end, the cleaned data consists of 109 features whereas the number of scored goals is the dependent variable. With the help of a backwards feature selector, the most important features are chosen before the data is fed into the models. The applied machine learning models are *random forest repressor, LASSO model*, and *KNeighbors*. The applied deep learning methods are *Feedforward Neural Network* and *Recurrent Neural Network*.

The following questions are answered within this project:

1. What method fits best to predict the number of goals of a football game?
2. What is the probability of x goals in the next game of two independent teams and how can you derive the winner from that?

To answer those questions, supervised machine learning techniques are applied, as the machine learns from labeled training data and a collection of training examples to infer a function.

2. Data Exploration

In this Section, the data used for this project is presented regarding how and where it was sourced, how it was applied, and how the data needs to be cleaned in order to get the best outcome of the applied machine learning methods.

2.1. Metadata

Today, there are multiple sports analysis suppliers available in the field of football analytics. In the first place, this kind of data is used by professional football teams to get information about past games in order to execute further analysis and to have numbers that can be compared over time. Additionally, this kind of data is used in the industry to scout players based on their past performance in order to find the skills needed in a given team. Often, such data suppliers have their own online tool where one can find all kinds of statistics for a single game – either for a team, a league, or a single player. For this project, the data was taken from *InStat* [4]. *InStat* provides professional football teams all around the world with data, which is in the same format and can be exported, for instance to a CSV file. *InStat*, as well as other data providers, obviously do not offer the data for free. Due to limited resources for this project, details on how to web scrap the data is not presented here. The data used was downloaded manually and the access was organized by a third party.

For this project, all features of the dataset were measured per 90 minutes for each team and game, for example the number of passes made by each team in 90 minutes of a game. This facilitated a comparison of the individual features from team to team. Mainly, the raw data consists of numeric variables. There are, however, also 27 non-numeric features, such as the information about whether the game is a home- or an away game. See Figure 1 for the details on the raw data format.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 755 entries, 0 to 754
Data columns (total 98 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Match date                              755 non-null    object
 1   Matches                                 755 non-null    object
 2   Team                                    755 non-null    object
 3   InStat Index                            755 non-null    int64
 4   Goals                                   755 non-null    int64
 5   Chances                                 755 non-null    int64
 6   Chances successful                      755 non-null    int64
 7   Fouls                                   755 non-null    int64
 8   Yellow cards                            755 non-null    int64
 9   Red cards                               755 non-null    int64
 10  Tactics                                 755 non-null    object
 11  Offsides                                755 non-null    int64
 12  Corner                                  755 non-null    int64
 13  Shots                                   755 non-null    int64
 14  Shots on target                         755 non-null    int64
 15  Passes                                  755 non-null    int64
 16  Accurate passes, %                      755 non-null    object
 17  Key passes                              755 non-null    int64
 18  Key passes accurate                     755 non-null    int64
 19  Crosses                                 755 non-null    int64
...
 96  Opponent's xG per goal                  755 non-null    float64
 97  home_Away                               755 non-null    object
dtypes: float64(11), int64(60), object(27)
memory usage: 578.2+ KB
```

Figure 1 - Information on the raw data used for this project

The available data also consists of information about the opponent's performance within a game. This data is identified by the column names that start by 'Opponent's […]'. Analyzing these factors are important because the opponent's performance obviously also affects the number of goals a team scores. This topic will be elaborated further in Section 3.6 Predicting the number of goals of this paper.

The chosen data starts from season 2020 and is updated after every new game until the end of the first round of the season 2022 / 2023. A season of a professional team in the Swiss football league consists of 36 games. Thus, in this project, the available data for a team range from 88 to 53 data rows.

The code is available on github as a python notebook file (.ipynb) and can be run on a local computer. The data is not available online, but can be accessed by contacting the author.

2.2 Data Flow

In the previous section, it was explained how the required data was extracted. In Figure 2, this first step is termed "Raw Data". After the data extraction, the retrieved data was visualized by plots and tables. The dependent variable, i.e., the number of goals, is in the center of interest in those plots and the objective is

to get an understanding of the distribution and dependencies of the independent variables towards the dependent one.

After that, the data was cleaned by removing the nonviable features and adding new columns (see Section 2.5 Data Cleaning). Then, the data was divided into training data and test data. Since the data is presented as a time series, the division was made according to the dates. The size of the test data is part of the model evaluation and is not fixed at all time. With this step, the data processing part ends.

After dividing the data into training and test data, the training data is fed into the feature selection method after which the different machine learning methods were applied to it. The following models were then examined by applying the training data to it (see Section 3.4 Applied Regression Models & Section 3.5 Deep Learning Models). Based on the accuracy of the training data (mean squared error of actuals goals vs. predicted goals), the hyperparameters of the models were fine-tuned to improve their accuracy. In the flow chart below (Figure 2), this step is visualized as "Performance Examination" with an arrow that traces back to "Feature Selection" and another one to "Final Model". Once the best fits are found, the final models are used for further calculations.

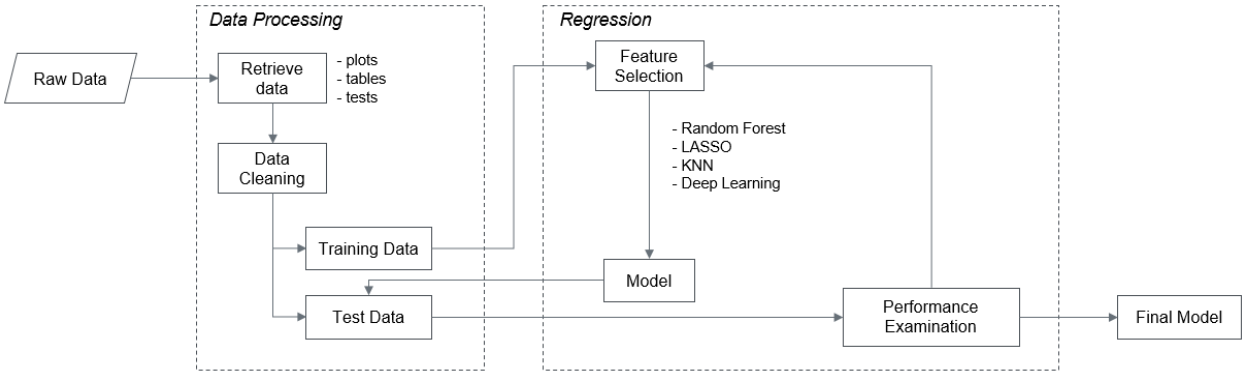

Figure 2 - Data Flow visualized

2.3 Data Plotting

For this section, the data was plotted in different graphs in order to get a better understanding of how the data distributes and correlates. The biggest challenge that was faced in this step was the high number of available features. As a start, all numeric features were plotted in a histogram to get a glimpse at their different distributions (see Figure 3). More of the plotted data can be found on github [5].

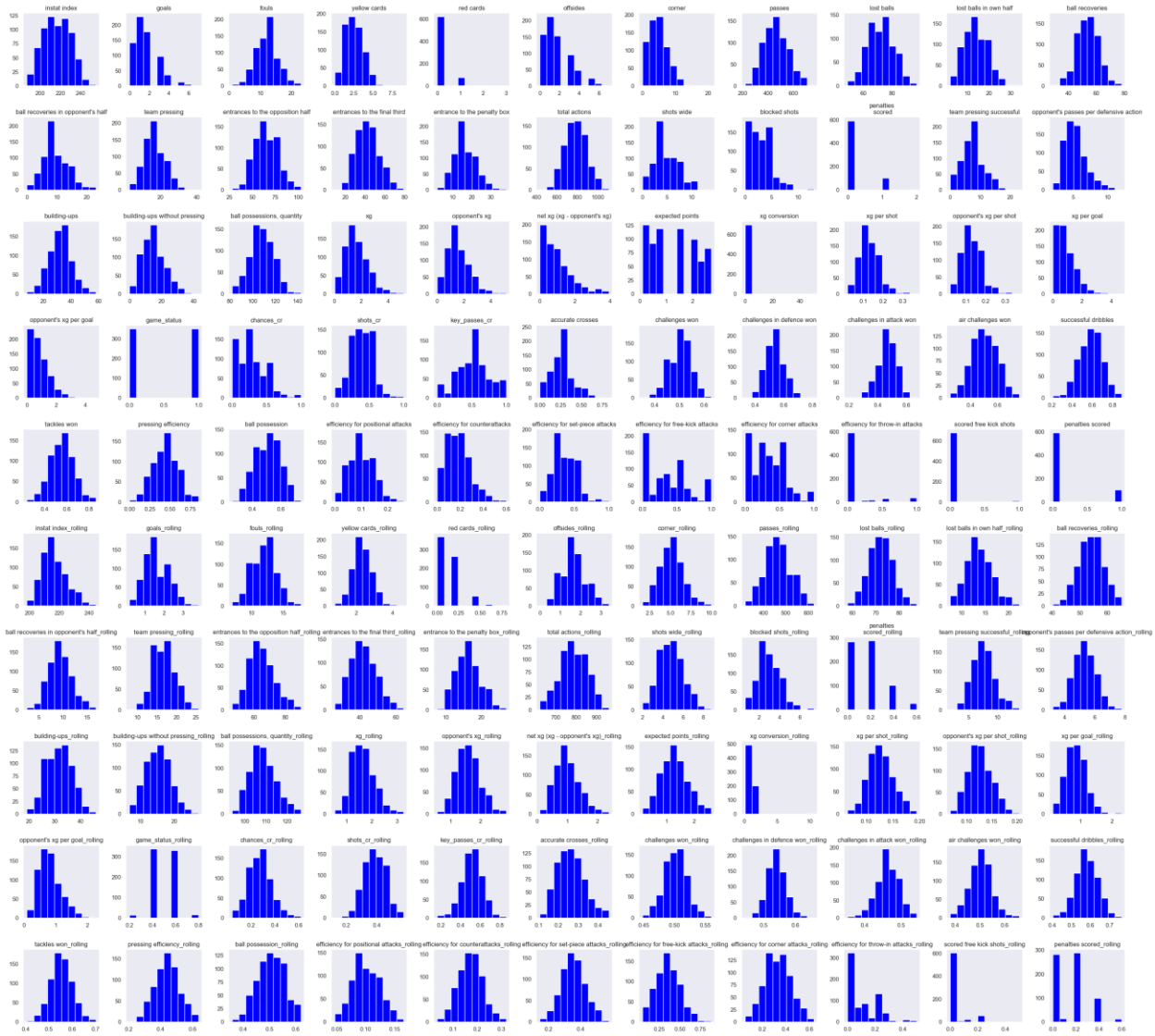

Figure 3 - Histogram of all numeric Variables

The distribution of the dependent variable, the number of scored goals, is the most anticipated. Figure 4 and Figure 5 illustrate the distribution of the goals and the 'xG', the expected goals for the teams available in the data set. The 'xG' is the theoretical expectation of number of goals per game, calculated by the data provider. It is assumed that this float number shows the same distribution as the actual goals.

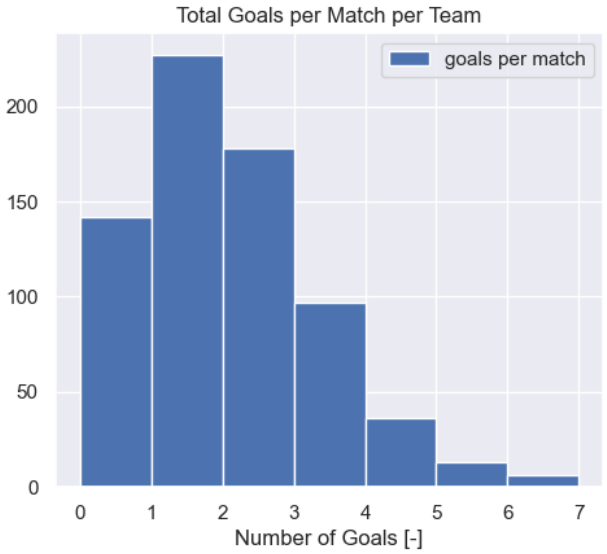

Figure 4 - Histogram for the goals in total and for all Teams

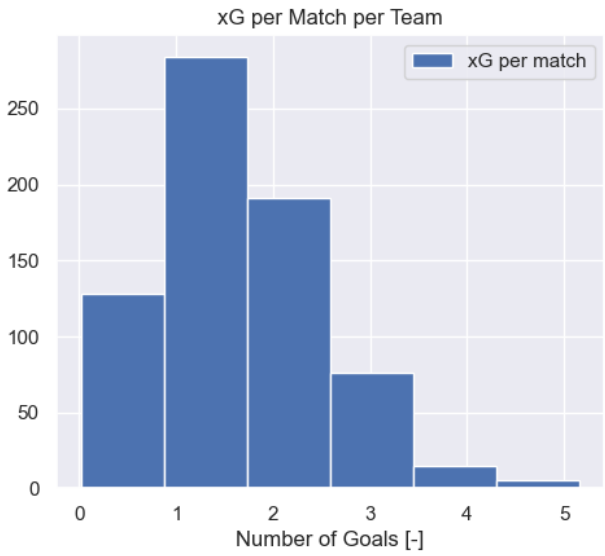

Figure 5 - Histogram for the 'xG' in total and for all Teams

The distribution of the goals is especially important for other parts of the project and will therefore be elaborated on in more detail. In order to do that, one can apply the Q-Q plot. A Q-Q plot is a scatter plot with two sets of quantiles against each other, see Figure 6 and Figure 7. If both sets of quantiles are coming from the same distribution, the points will form a nearly straight line and therefore a normal distribution is assumed [6]. This seems not to be the case for neither the scored goals nor the 'xG' value.

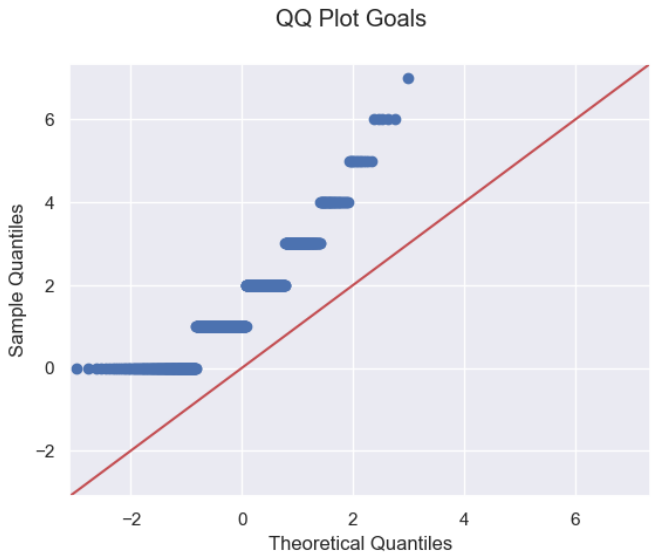

Figure 6 – Q-Q Plot Goals

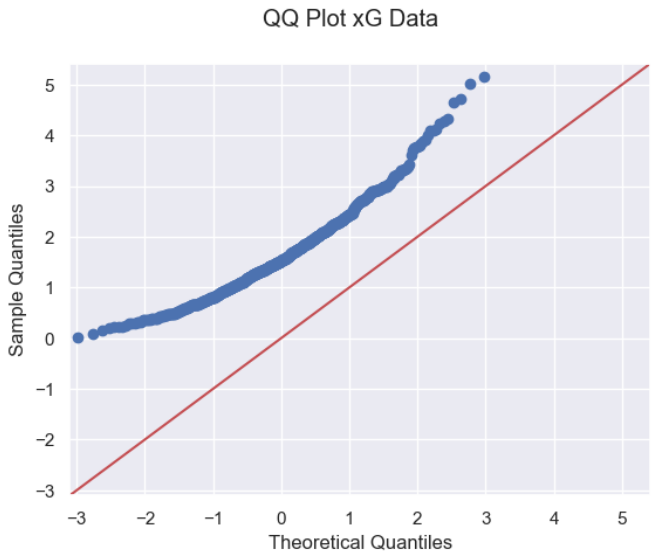

Figure 7 - Q-Q Plot 'xG'

The boxplot graph in Figure 8 further shows that the goals and the ‘xG’ do not follow a normal distribution. Boxplot is not only used for detecting outliers of the data but to visualize and understand its distribution. In Figure 8, the features are looking skewed as the distance between both whiskers is not equal and hence not normally distributed.

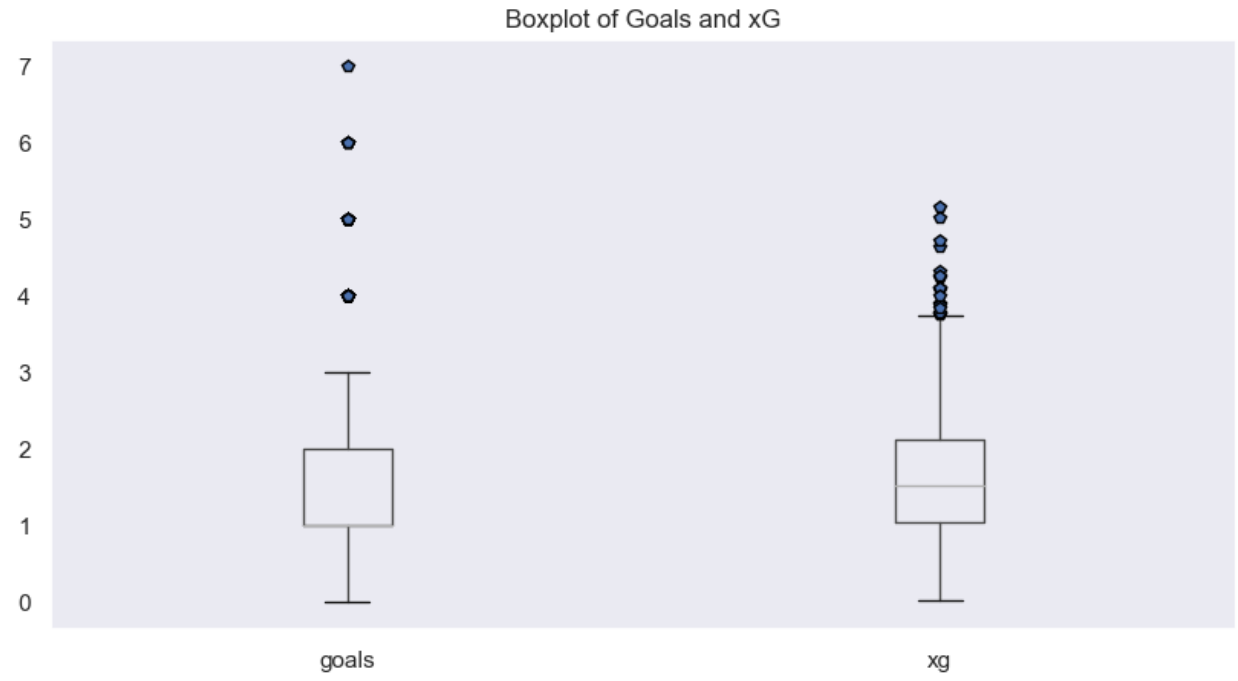

Figure 8 - Boxplot of Goals and 'xG'

As an alternative to employing visual representations for understanding distribution, statistical tests can be applied in order to find out whether the data is normally distributed or not. One possibility to check this is the *D'Agostino's K-squared test*. This test aims at assessing the compatibility of given data with the null hypothesis that the data is a realization of independent, identically distributed Gaussian random variables [7].

The *D'Agostino's K-squared test* is appleid to the data, in specific to the goals and the 'xg', see Figure 9. The null hypothesis is rejected when the p-value is < 0.05, which is given here in this case.

```
# D'Agostino's K-squared test

print(stats.normaltest(df_cleaned['goals']))
print(stats.normaltest(df_cleaned['xg']))
✓ 0.1s
NormaltestResult(statistic=73.8727052268367, pvalue=9.093810047217652e-17)
NormaltestResult(statistic=77.1388585470318, pvalue=1.7762729261244642e-17)
```

Figure 9 - Printscreen of 'D'Agostino's K-squared test' applied on the goals and 'xG'

2.4 Data Quality

The data used for this project is of very high quality, as it comes from one (trusted) source, has one format, is easy to access and easy to extract. The data itself is trustworthy as the same data is sold to professional football clubs. Today, this kind of data is collected with technical support, such as special located cameras or shirts, which the footballers are wearing during the games. The majority of features are either integers

or already a ratio between two features, as for example 'Low pressing, %', which is the ratio between 'Low pressing' and 'Low pressing successful'.

Not easy to handle, however, is the record size of the available data and the fast-changing team performance of a football club from one season to another or even from one week to another. In addition, the impact of individual players and coaches, which change frequently, affects the performance of a team. Also, other independent variables such as sponsors, weather, etc., are currently not considered in the dataset. Thus, one has to be careful when choosing the right features and models.

There is a lot of data available for individual players as well. However, this kind of information has not been part of this project yet, i.e., only team statistics were considered without considering the impact of a single player. Nevertheless, it can be interesting to see how the model would react if player's data is included in the model.

2.5 Data Cleaning

In this Section, it is elaborated how the raw data is cleaned in order to get a data set that can be further applied to machine learning methods.

2.5.1 Reformatting, Renaming, and Dropping Columns

The data is loaded in a CSV file format. As the data is represented as a time series, one has to ensure that the date column has the right format. In this case, this is done by applying the function "pandas.to_datetime" as it converts a scalar, array-like, Series or DataFrame/dict-like to a pandas date time object [8]. After that, the column 'Home_Away', which gives information on whether a game was played at home or away for a team, is converted into a categorical column named 'Games_Status', with 0 = home game and 1 = away game.

As shown in Section 2.3Data Plotting, some columns are in direct relation to each other while others already exist in a ratio / percentage format. To adequately make use of the first type, the according ratios of those columns must be calculated and added as a new column. For example, 'Chances successful' and 'Chances' are only representative if they are used as a ration of the successful chance and the total chances. In total, three such ratios are manually calculated and added to the Dataframe, see Figure 10. As mentioned before, other columns, such as 'Successful dribbles, %', are already listed as percentage numbers. To use them in the Dataframe, however, they must be converted to float number format.

```
#convert data to percent to have compareable dta format with other already existing ratios
df_all['chances_cr']=df_all['Chances successful']/df_all['Chances']
df_all['shots_cr']=df_all['Shots on target']/df_all['Shots']
df_all['key_passes_cr']=df_all['Key passes accurate']/df_all['Key passes']
```

Figure 10 - Overview of Ratios that are manually built and added as new features

In reverse, the columns which are used to build such ratios are dropped from the Dataframe. Additional to those columns, also non-numerical columns such as ‘Matches’ and ‘Tactics’ or ‘Ball possession, sec’ are dropped from the Dataframe.

In order to use the Dataframe for further analysis and predicting the number of goals for future games, the column names are all renamed with lower caps. Besides features which are already translated into ratios, there are also many columns consisting of integers.

2.5.2 Adding Rolling Averages

Rolling averages (also known as moving average or running average) are often used in time series data to help filter out the noise and create smooth curves for past data points. In this project, it is used to identify short-term trends. Thus, with the Pandas function ‘.rolling’, a rolling average of the last five games is calculated for every numeric feature [9].

To do so, one has to ensure that the rolling averages are applied to the single teams only and not to random consecutive games of the dataset. That is why a function is defined to calculate the rolling averages for a single team first. After, this function is applied to all other teams within the data set. The newly added columns are identified with the addendum of ‘_rolling’ to the original column name, e.g. ‘Ball possession_rolling’. One has to consider, depending on how many past games your rolling averages is calculated from, the same number of (oldest) data rows is excluded from the Dataframe. In the end, including the new and non – numeric columns, the cleaned data set consists of 112 rows.

2.6 Test / Training Set

Once the data is cleaned, the Dataframe is divided into a test and a training Dataframe for each individual team of the data set. As will be discussed in Section 3.1 Problem Statement, for every team within the data set, a single model is calculated. Thus, a separate Dataframe must be saved in advance for every team. The data sets are identified by the addendum of the short name for the respective team at the end of the saved file.

Instead of applying a steady ratio, as for example 80/20, for dividing the Dataframe into training and test data, the data is split by selecting the latest x games of the full data set as the test data, and the rest (total games – x latest games) as the training data.

2.7 Data Scaling

As explained in Section 2.5 Data Cleaning, the individual Dataframes consist of features with different value formats. On the one hand, the columns represent a ration of two values and on the other, columns are shown in absolute values, e.g. number of passes per game. Thus, in order to balance the impact of all

variables on the calculation and to improve the performance of the algorithm, the data is scaled after dividing it into training and test data sets. To do so, the *MinMaxScaler* from sklearn is applied to the data. This estimator scales and translates each (numeric) feature individually so that it is in the given range on the training set, for example between zero and one [10]. The same model is then applied to the test set.

3. Machine Learning Methods

In this section, five different applied machine learning methods are presented and it is discussed why they were chosen for the regression problem at hand. Additionally, to compare the outcome of the different methods even better, the approach on how to use these models to predict next games, for which data is not yet available, will be explained.

3.1 Problem Statement

The objective of this project is to find machine learning regression models predicting the number of goals a soccer team will score in the next game against a known opponent. With this predicted number of goals, new calculations are triggered, such as the probability of which team is going to win the game and the probability of how many goals are going to happen during the game. Scored goals of football games are Poisson distributed, see Section also 3.6 Predicting the number of goals. Thus, the Poisson distribution is the way of how to predict these probabilities. Such 'task driven approaches' are typically employed for supervised machine learning algorithms.

As explained above, the data used for this project consists of information about different football teams. In order to get the best-fitted models per team, a separate model is generated for each team instead of one general model that should fit all. The division of the data into its individual teams happens after the data cleaning (see also Section 2.5 Data Cleaning). Like this, one ensures that every team has the same data format and source a comparison of the model outcomes is fair between the teams. The downside of this approach is the amount of available data per team. In Section 4 the impact of the available data sizes is discussed.

For more details on how the multiple Dataframes of the different teams are manipulated simultaneously, one can refer to the code on github [5].

3.2 Libraries

Without the access to libraries from different providers, applied data science would not be possible. The following libraries were used for this project and the code. For more details on the code itself, see [5].

pandas: pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures, and data analysis tools for the Python programming language [11].

sklearn: Scikit-learn is an open-source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities [12]. The following tools (and their available methods) are applied from *sklearn* in this project:

- MinMaxScaler
- RandomForestRegressor
- SequentialFeatureSelector
- RidgeCV
- KneighborsRegressor
- linear_model.Lasso

matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible [13]. Following tools are used in the project, to plot the data (see Section 2.3 Data Plotting):

- pyplot
- seaborn

NumPy: NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [14]. In this project it is mainly used to reshape arrays so the data fits as an input for the deep learning methods.

SciPy: SciPy is an open-source software for mathematics, science, and engineering [15]. In this project, following tool from numpy is used to apply the predicted goals to Poisson distribution:

- scipy.stats

TensorFlow: TensorFlow makes it easy for beginners and experts to create machine learning models [16]. In this project it is mainly applied in the deep learning part, see Section 3.5 Deep Learning Models. Following tools have been applied from TensorFlow:

- keras

3.3 Feature Selection

A Sequential Feature Selector adds ("forward selection") or removes ("backward selection") features to form a feature subset in a greedy fashion. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator. In the case of supervised learning, the Sequential Feature Selector looks at the features (X) and the desired outputs (y) [17].

For this project, the backward selection approach was chosen. Thus, the Sequential Feature Selector starts with 109 features (X) and removes the number of the features until a certain value is reached. This value, '*n_features_to_select'*, is either set to 'auto', to an integer or to a float. When 'auto' is chosen, the Sequential Feature Selector removes features until the score improvement does not exceed a certain tolerance. Is this tolerance = *None*, half of the features are selected. When an integer is set, the Sequential Feature Selector stops when this number of features is reached. If float between 0 and 1 is set, the Sequential Feature Selector stops when this fraction of features is reached. The number of selected features and its impact on the model performance is discussed in Section 4.1 Valuation Feature Selector.

Every Sequential Feature Selector needs an unfitted estimator at the beginning. For this project, it is a *RidgeCV*. RidgeCV is a cross validation method in ridge regression. Ridge regression is part of the regression family which uses L2 regularization and adds "*squared magnitude*" of coefficient as penalty term to the loss function [18].

The *alpha* value of this estimator can be manually set. The *alphas* is an array of alpha values to try by the estimator. By default, this array is 0.1, 1.0, 10.0. It can be seen as the regularization strength that improves the conditioning of the problem and reduces the variance of the estimates [19].

One reason this RidgeCV estimator was chosen can be seen in the flow chart in Figure 11, offered by scikit-learn.

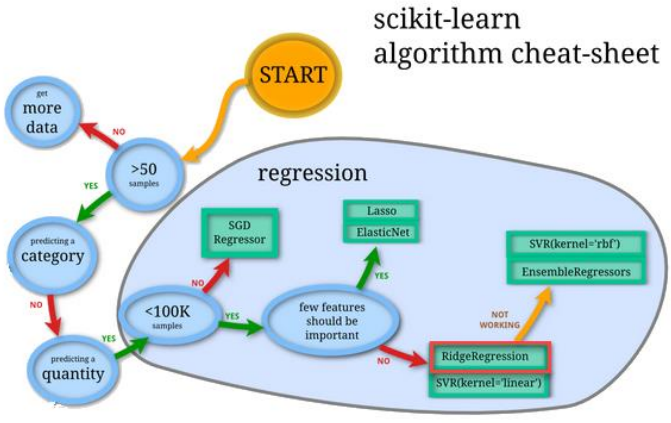

Figure 11 - Algorithm cheat - sheet from scikit – learn [20]

3.4 Applied Regression Models

In the following subsections, the applied regression machine learning methods used for this project will be explained.

3.4.1 Random Forest Regression

The *Random Forest Regressor* is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [21]. In the regression case, the decision tree starts with the root of the tree and follows divisions based on variable outcomes until a certain leaf node is reached.

One hyperparameter of this Random Forest Regressor is the value for the number of trees in the forest (*n_estimators*), whereas the default number is 100 in the latest version of the model. The maximum depth of the tree (*max_depth*) is either dependent on a manual integer input or expands until all leaves contain less than the minimum number of samples required to divide an internal node. In our case, the *bootstrap* is set to *True,* indicating that bootstrap samples are used when building trees. The chosen *criterion* (= function to measure the quality of the split) is 'squared-error' for the mean squared error. The *random_stat* controls both the randomness of the bootstrapping of the samples used when building trees (if *bootstrap* = True) and the sampling of the features to consider when looking for the best split at each node (if *max_features* < *n_features*). It is set to 42 in order to keep the result stable across a number of different distinct random seeds.

The Random Forest Regressor from sklearn offers multiple methods to apply. In our case, the *.fit* method, to build a forest of trees from the training set, the *.score* method, to return the coefficient of determination of the prediction (MSE) and the *.predict* method, to predict regression target for the features (=number of goals), is applied. The score method and the regression method are applied to the training and test data set in order to evaluate the model.

3.4.2 LASSO Model

LASSO regressions are well known techniques which are typically used for building learning models in presence of a large number of features, which is due to their capability to prevent overfitting and reduce the complexity of the model. The LASSO (Least Absolute Shrinkage and Selection Operator) regression model uses L1 regularization technique that uses shrinkage, which penalizes 'absolute value of magnitude of coefficients' (L1 penalty). As a result, LASSO appears to render coefficients to absolute zero. Additionally, LASSO regression is useful to obtain a subset of predictors by eliminating less important features [22].

The sklearn model has different hyperparameters which can be manually set. The parameter *alpha* is set by default to 1.0, and is a constant that multiplies the L1 term and controls the regularization strength. It must be a non-negative float and when set to zero, the objective is equivalent to ordinary least squares, solved by the Linear Regression object [23]. Also, the *random_state* and the *max_inter* parameter were set manually for this project. The LASSO regression has the same methods in place as described before for the Random Forest Regressor in Section 3.4.1 Random Forest Regression. Thus, for the LASSO regression the fit, predict, and score methods are applied in this report.

3.4.3 Kneighbors Regressor

Sklearn also provides the *Kneighbors Regressor* model. Kneighbors Regressor, also known as KNN or k nearest neighbors, is a simple algorithm that stores all available cases and predicts the numerical target based on a similarity measure (e.g., distance functions) [24]. A non-parametric method (meaning, it does not have strict requirements on the shape and distribution of the data) approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood. Therefore, for a regression problem which predicts the actual numerical value of a new sample, the algorithm just takes the mean of the nearest *k* neighbors [25]. In the end, the value of *k* (or *n_neighbors*) is the parameter which has to be set manually and its optimal value depends on the available data. One can say that the smaller this value *k*, the bigger the overfitting. On the other side, choosing a large value will lead to underfitting and will be computationally expensive.

3.5 Deep Learning Models

In this Section, the applied deep learning models are explained in more detail. Both models are provided by Tensorflow, see also Section 3.2 Libraries.

3.5.1 FNN – Feedforward Neural Network

In *Feedforward Neural Networks* (FNNs), the information flows in only one direction: forward. More specifically, it flows from the input layer through the hidden layers to the output layer, but never backwards in feedback loops. The FNN multiplies a matrix of weight factors with the inputs and generates the outputs from these weighted inputs. FFNs do not retain a memory of the inputs they have processed [26].

For this project, a FNN with multiple hidden layers was built and evaluated. Linear and ‘relu’ were chosen as activation functions. The units / number of neurons in each layer were manually reduced from one layer to another, starting from 109 (total number of features) and ending with one unit in the output layer. The model is set up with up to three hidden layers. How that makes sense will be explained in Section 4.2.2 Setting 2. *Adam* is chosen as an optimizer and the loss function (function that compares the target

and predicted output values) is mean squared error. As for regression problems, the metric 'accuracy' (= number of correct predictions) makes no sense, the model will be evaluated based on the loss function.

The settings of batch size and optimal epochs number will be discussed in Section 4.2 Valuation of applied Machine learning models.

3.5.2 RNN – Recurrent Neural Network

Recurrent Neural Networks (RNNs) have shown to achieve the state-of-the-art results in many applications with time series or sequential data [27]. RNNs are a type of neural network that retains a memory of what it has already processed and thus can learn from previous iterations during its training. An RNN makes recurrent connections by going through *temporal feedback loops*: the output of a preceding step is used as an input for the current process step.

A normal RNN is composed of three elementary components: the input layer, the hidden layers, and the output layer. Each layer consists of so-called *nodes* (aka *neurons*). The hidden layers in an RNN do not only produce an output, but they also feed it back ("backpropagate" it) as an input for training the hidden layer on the next observation. An input value, when it is passed from its node on one layer to a node in another layer, travels along an *edge* (the connecting line) between the nodes. The receiving node sums up all the inputs it receives to a *total net input* and feeds this net input into an activation function to compute the output.

Among the most frequent used activation functions are the logistic or sigmoid function; the step or heaviside function (comparable to a Boolean or binary yes/no decision); the hyperbolic tangent function (tanh); and the ReLU function (rectified linear unit, max(0,x)). Nonlinear activation functions like the logistic or hyperbolic tangent functions help the network to adapt itself to nonlinear problems when mapping the input to the training output [26]. For a regression problem, a linear activation function is suggested.

The input layer must have the shape *number of input sample, number of time steps (=number of samples in the past that are considered), number of features.* For the training set, the *number of input sample* would be the training data set size minus the *number of time steps* (same logic for the test set), as those time steps are needed to predict the next outcome in the future and thus fall off the data set at the beginning. For this project, the RNN was built with two hidden layers, both using the activation function 'relu', followed by a dropout layer and a flatten layer. Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term 'dropout' refers to dropping out units (both hidden and visible) in a neural network [28], [29]. The flatten layer is needed, since the dense layer (followed after the

last hidden layer) cannot process a matrix and has to be a vector. What the flatten layer does is basically just to open up the 2D matrix and represent it as a 1D vector [30].

After the flatten layer, a fully connected layer (dense layer) is set and the model ends with an output layer. The output layer has a linear activation function. The *units* per layer are manually adjusted and its impact on the outcome of the model is discussed in Section 4.2.4 RNN. *Return_sequences,* whether to return the last output in the output sequence or the full sequence, are by default set to 'False' and are chosen to be 'True' in the hidden layers. That is why the flatten layer was added to the model as well.

Adam is chosen as an optimizer and the loss function (function that compares the target and predicted output values) is mean squared error. As for regression problems, the metric 'accuracy' (= number of correct predictions) makes no sense.

The settings of batch size and optimal epochs number will be discussed in Section 4.2.4 RNN.

3.6 Predicting the number of goals

The objective of the models presented in Sections 3.1 Problem Statement, is to predict the number of goals of a next match between two teams. To do so, per team, the mean of the selected features from the last x games is calculated and added as a new data record to the test set. This added data record is fed to the model and with the method *.predict,* the number of goals for the next game is calculated. Since the factor 'home games or away games' can have an impact on a team's performance, the mean of the required features is calculated by filtering the data set on either home or away games. The optimal number of games that should be considered from the past to calculate the future will be elaborated in Section 4.3 Valuation of predicted goals. As the required features can also include opponents' information, depending on the selected features, you have to ensure that the information of the next opponent is considered and added to the test set as well.

Once the predicted number of goals of the next game is simulated, one can use those values to calculate the probability of which team is going to win the game and how many goals to expect in total in the corresponding game. This is done by making use of the Poisson distribution. A variable follows a Poisson distribution when the following conditions are true:

- Data are counts of events.
- All events are independent.
- The average rate of occurrence does not change during the period of interest.

The numbers of goals are count data that happen during 90 minutes (or the full length of the game). The count data have discrete values comprised of non-negative integers (0, 1, 2, 3, etc.), and their distributions are frequently skewed (see also Section 2.3 Data Plotting).

This discrete probability distribution expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant mean rate and independently of the time since the last event. The mathematical expression looks like the following:

$$f(k;\lambda) = \Pr(X{=}k) = \frac{\lambda^k e^{-\lambda}}{k!},$$

where

- k is the number of occurrences ($k = 0, 1, 2, \ldots$)
- e is Euler's number ($e = 2.71828\ldots$)
- ! is the factorial function.

The positive real number λ is equal to the expected value of X and also to its variance. In our case, λ is the predicted number of goals, calculated by the applied models. Now, to calculate the probabilities of who is going to win the game, based on this value λ, one sums up the probabilities of every possible win of the home team (p_w_home) and every possible draw (p_draw). Then, the possibility of the away team to win is 1 - p_w_home - p_draw.

The number of total expected goals per game is calculated by considering the λ of the home and away teams and by calculating the probability of 1, 2, 3, and so on goals, by again applying the Poisson distribution formula.

4. Results

In this chapter, the models and approaches, explained in Chapter 3, are evaluated and compared with each other. The comparison between the models is based on the *score* value and the actual predicted goals of the next game, for which no data is available yet.

The data for the 'Swiss Super League' (highest professional football league in Switzerland) will be used as the example data set. The Swiss league consists of 10 teams and one season has 36 games. Per team, the dataset consists of 88 data rows (depending on how long the team plays in this league) and starts from the season 2020 / 2021. It is also a part of this chapter to elaborate on the number of data records and their impact on the performance of the models applied.

4.1 Valuation Feature Selector

For more details on the set up of the applied Sequential Feature Selector, see Section 3.3 Feature Selection.

As a starting point, the number of features to select (*n_features_to_select)* is set to 30 (the model starts with 109 features). In total, 105 different features are selected by the model. That is, only four features have not been selected at all for at least one team. The computational time for nine teams is 22 minutes and 40 seconds / ~ 2.5 minutes per team. The lower the number of features to select the longer the computational time. This makes sense since we are applying the backwards wrapping method on the Sequential feature Selector. The impact of *n_features_to_select* on the model performance is elaborated in more detail in the next Section.

Although 104 features are selected by nine teams, interestingly, it is a different combination of features for every team. However, there are also features which are selected by every team. Table 1 lists the features selected for at least four teams, for *n_features_to_select* = 30. 'Count' means the number of teams the according feature was selected for by the model.

Selected Features	Count
'xg'	9
'chances_cr'	9
'xg conversion'	8
'instat index'	7
'efficiency for throw-in attacks_rolling'	5
'efficiency for counterattacks'	5
'shots_cr'	5
'xg per goal'	5
'efficiency for positional attacks'	5
'net xg (xg - opponent's xg)'	5
'opponent's xg per shot'	5
'lost balls in own half_rolling'	5
'key_passes_cr'	5

Table 1 - List of counts (> 5) of selected features (n=30)

As visible in Table 1, 'xg' and 'chances_cr' are selected for every team. This makes sense as the 'xg' (expected goals) consists of the data of the theoretical expected goals of a certain game (calculated by the data provider) and 'chances_cr' is the ratio of successful chances and number of chances. Both features that have a direct dependency on the number of scored goals and features you possibly would also nominate naturally, without the help of a computer. Additionally, for 7 out of 9 teams, the feature 'instat index' is selected. 'instat index' is a value that summarizes the performance of a team per game and is an invention from the data provider. Generally, one can say, the higher the 'instat index' the better performed the team. In other words, the 'instat index' might be an indication on how many goals a team scored. Again, also the selection of this feature makes common football sense.

Whenever a feature name starts with 'opponent's*', it indicates that data from the opponent team also has an important impact on the scored goals. Every team has at least one such feature selected. Also, the '*_rolling' features are worth to mentioned, see also Section 2.5.2 Adding Rolling Averages. As every team has at least one such manually added feature selected, shows their importance on the outcome of the model and confirms their eligibility in the data set.

4.2 Valuation of applied Machine learning models

In this Section, the models described in Section 3.4 Applied Regression Models are evaluated and compared between each other. Only the best hyperparameter setting are listed here. The results are discussed in the next Sections and for more details on the models, see github directory [5].

The hyperparameters are found by trial and error, mostly with the approach of starting with high values and reducing them steadily until the optimum is found. The models are compared based on their *score* value on the training and test set, which is for all the mean square error.

4.2.1 Setting 1

Number of selected features: 30

Training and test split: 12 latest games considered as test set

For the Random Forest Regression model, few teams have a very low *score* value on the test set (< 50%) at the beginning, mostly with default hyperparameter values (*n_estimator = 100, max_dpeth = 10)*. This value can be improved, for example for the team Lugano (LUG), by adjusting the *max_depth* to a lower number. However, this then decreases the *score* value for other teams.

As an example, following the plots of predicted vs. actual goals for the teams Lugano and St. Gallen (Figure 12 & Figure 13), both teams with the lowest test *score* value. In the end, the best average setting, considering the *score* value on the test set, is found when with *n_estimator = 75* and *max_depth = None.*

Figure 12 - Predicted vs actual Goals Lugano, RF model

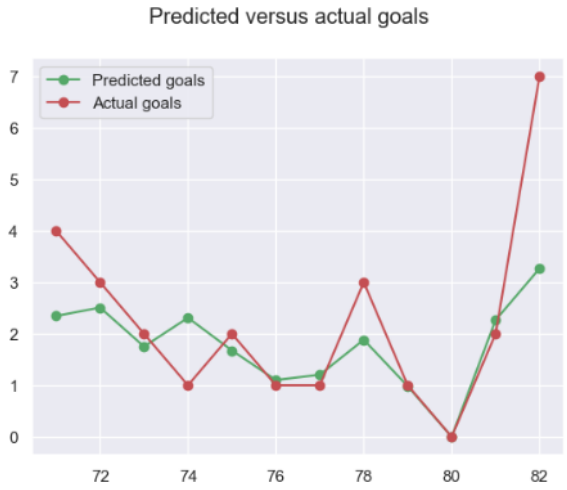

Figure 13 - Predicted vs actual Goals St. Gallen, RF model

It gets visible that especially the score value of St. Gallen is biased by the last game, where the actual goal can be considered an outlier. For Lugano on the other side, the situation is a bit more complex. It seems like the team underperformed in average compared on the test set to what the model expected.

The best score value on the test set has Basel (see Figure 16). As an example, following the plots of the teams Basel and Sion, for the described setting (see Figure 14 & Figure 15).

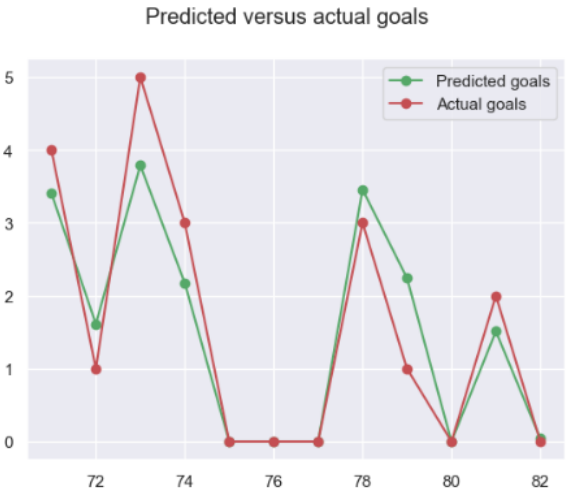

Figure 14 - Predicted vs actual Goals Basel, RF model

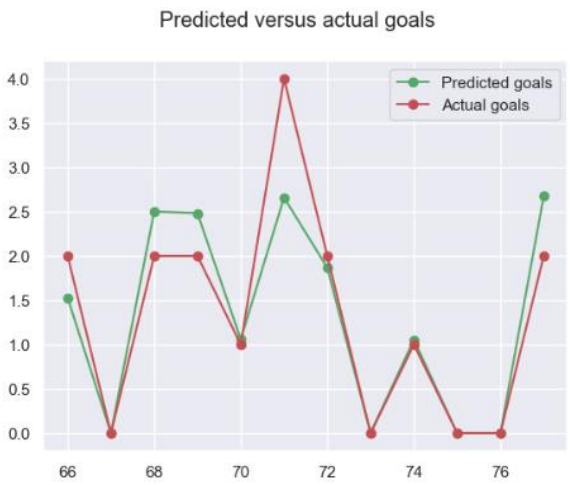

Figure 15 - Predicted vs actual Goals Sion, RF model

For the LASSO model, the best outcome is achieved with a relatively low *alpha* value of 0.01. The lowest *score* value on the test set is calculated for the team Grasshoppers with 52%. In general, the *score* value for the LASSO model on the training set is lower than for the random forest model. However, when considering the gap / ratio of the *score* value of the test set and training set, the LASSO model performs better. The LASSO model also tends to perform better on test sets the random forest model was struggling with, see Figure 16, e.g. team Lugano.

For the KNN, the best average *score* values for the training set is achieved with *n_neighbors = 3*. However, the *score* values, for test and training sets, are much lower than for the random forest model and the LASSO model. In following Figure 16, the best *score* values for the test and training set are highlighted.

	team	Train_Score RF	Test_Score RF	Train_Score_Lasso	Test_Score_Lasso	Train_Score_KNN	Test_Score_KNN
0	BAS	0.967680	0.896159	0.901764	0.690155	0.719426	0.163087
1	GRA	0.938649	0.593361	0.926314	0.528795	0.750882	0.360731
2	LUG	0.974881	0.171314	0.877442	0.699628	0.808331	0.095238
3	LUZ	0.966662	0.598672	0.818615	0.652276	0.557432	0.378788
4	SER	0.973183	0.832651	0.914543	0.848708	0.721874	0.293333
5	SG	0.969123	0.485862	0.934819	0.660054	0.765224	0.148874
6	SIO	0.980578	0.828107	0.919460	0.793668	0.727671	0.500000
7	YB	0.972383	0.804717	0.926426	0.829954	0.816286	0.574074
8	ZUR	0.972451	0.849904	0.901666	0.850428	0.652989	0.724395

Figure 16 - Score values for test set = 12 games, n_features_selected = 30

Training and test split: 20 latest games considered as test set.

In Figure 17, the best *score* values when expanding the test set to the latest 20 games are listed and highlighted for the different models.

	team	Train_Score RF	Test_Score RF	Train_Score_Lasso	Test_Score_Lasso	Train_Score_KNN	Test_Score_KNN
0	BAS	0.964205	0.870146	0.907212	0.634692	0.666359	0.128086
1	GRA	0.927717	0.510858	0.943319	0.172358	0.650000	0.224537
2	LUG	0.976344	0.435622	0.871250	0.600516	0.841466	0.294294
3	LUZ	0.964911	0.629634	0.813724	0.653644	0.475349	0.444444
4	SER	0.966118	0.797476	0.931463	0.764945	0.696309	0.290123
5	SG	0.970032	0.572474	0.937746	0.714767	0.769256	0.012598
6	SIO	0.964854	0.807741	0.919242	0.812627	0.639032	0.412516
7	YB	0.970226	0.845362	0.925736	0.804674	0.823866	0.546926
8	ZUR	0.969398	0.832418	0.910811	0.614924	0.574324	0.565657

Figure 17 - Score values for test set = 20 games, n_features_selected = 30

Training and test split: 8 latest games considered as test set.

In Figure 18, the best *score* values when reducing the test set to the latest 8 games are listed and highlighted for the different models. In average, the values for the test set are lower than for the test set with 20 data records but higher than for 12 data records.

	team	Train_Score RF	Test_Score RF	Train_Score_Lasso	Test_Score_Lasso	Train_Score_KNN	Test_Score_KNN
0	BAS	0.968367	0.787602	0.905927	-0.347904	0.743220	-0.286550
1	GRA	0.945405	0.601032	0.894965	0.708572	0.730732	0.577061
2	LUG	0.968532	0.221859	0.870342	0.630650	0.805430	0.497585
3	LUZ	0.964099	0.845554	0.818914	0.719224	0.624035	0.415415
4	SER	0.973927	0.966156	0.912612	0.931520	0.723209	0.361111
5	SG	0.972696	0.516461	0.929993	0.689622	0.757460	0.168568
6	SIO	0.978240	0.808869	0.916010	0.763785	0.759774	0.218107
7	YB	0.971522	0.864327	0.926111	0.861112	0.827714	0.626517
8	ZUR	0.973959	0.882770	0.894601	0.880597	0.658994	0.654800

Figure 18 - Score values for test set = 8 games, n_features_selected = 30

4.2.2 Setting 2

Number of selected features: 15

Training and test split: 12 latest games considered as test set

The setting for the hyperparameter are kept the same as for the Setting 1, see previous Section. Figure 19 shows the best *score* values for 15 features selected and a test size of the 12 latest games.

	team	Train_Score RF	Test_Score RF	Train_Score_Lasso	Test_Score_Lasso	Train_Score_KNN	Test_Score_KNN
0	BAS	0.966328	0.894530	0.891121	0.722391	0.744858	0.789976
1	GRA	0.948772	0.614013	0.900099	0.560103	0.872134	0.494673
2	LUG	0.978144	0.457829	0.873779	0.691409	0.859872	0.309524
3	LUZ	0.963554	0.709610	0.795924	0.645018	0.620451	0.242424
4	SG	0.978439	0.616521	0.918567	0.683558	0.837942	0.029775
5	SIO	0.976973	0.890571	0.849626	0.759348	0.705520	0.586667
6	YB	0.974747	0.848658	0.890312	0.793379	0.687768	0.644444
7	ZUR	0.979084	0.866816	0.834081	0.947316	0.836806	-0.124767

Figure 19 - Score values for test set = 12 games, n_features_selected = 15

In general, one can say the random forest model improves its *score* slightly compared to the model when selecting 30 features. In addition, teams that have a poor performance on the test set improved by more than double the value, see Lugano for example, compared to Setting 1. However, there are still few teams which perform better on the test set with LASSO model instead of the random forest model.

Training and test split: 20 latest games considered as test set

When expanding the test set to consider the last 20 games, the *score* values change accordingly. In Figure 20 those calculated values are listed. So far, the best *score* values in average with these settings of test set data size and number of selected features. This is also visible when plotting the actual vs. predicted goals, see

	team	Train_Score RF	Test_Score RF	Train_Score_Lasso	Test_Score_Lasso	Train_Score_KNN	Test_Score_KNN
0	BAS	0.964155	0.899259	0.898214	0.644782	0.743068	0.647634
1	GRA	0.923867	0.559241	0.901070	0.416395	0.794444	0.452160
2	LUG	0.975733	0.536024	0.864002	0.622668	0.877588	0.061562
3	LUZ	0.965036	0.703830	0.805174	0.628321	0.583416	0.261518
4	SG	0.973508	0.687598	0.919579	0.713780	0.837794	-0.080468
5	SIO	0.969689	0.917977	0.846410	0.765252	0.718511	0.457216
6	YB	0.969751	0.853208	0.891031	0.764239	0.673661	0.743797
7	ZUR	0.976293	0.858311	0.845734	0.713053	0.825906	0.116162

Figure 20 - Score values for test set = 20 games, n_features_selected = 15

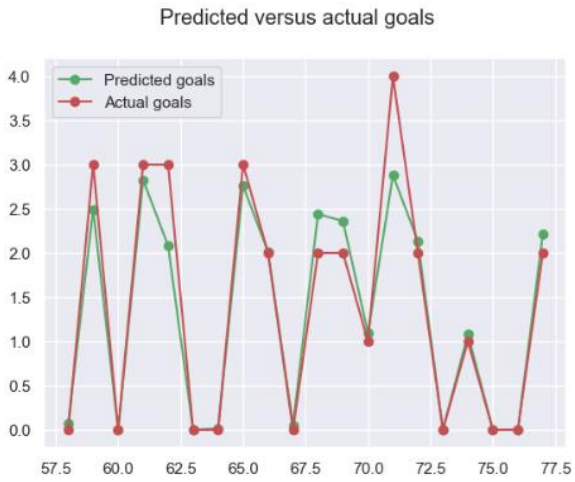

Figure 21 - Predicted vs. actual goals Sion, RF model, n_features = 15, test size = 20 games

Figure 22 - Predicted vs. actual goals Lugano, RF model, n_features = 15, test size = 20 games

Training and test split: 8 latest games considered as test set

When expanding the test set to consider the latest 8 games, the calculated *score* values impacted accordingly. In Figure 23 the best values are listed and highlighted. In average, the test *score* value decreases compared to the previous test set size.

	team	Train_Score RF	Test_Score RF	Train_Score_Lasso	Test_Score_Lasso	Train_Score_KNN	Test_Score_KNN
0	BAS	0.968209	0.736384	0.898852	-0.307883	0.779591	0.614035
1	GRA	0.945949	0.588611	0.881625	0.668530	0.805877	0.591398
2	LUG	0.970192	0.368039	0.867707	0.619066	0.850935	0.072464
3	LUZ	0.957988	0.819967	0.780325	0.683278	0.646216	0.503504
4	SG	0.976254	0.724522	0.908598	0.745865	0.832831	0.013097
5	SIO	0.974425	0.940978	0.841863	0.787716	0.718269	0.670782
6	YB	0.975717	0.848575	0.888600	0.817951	0.723748	0.738562
7	ZUR	0.977667	0.902681	0.834483	0.942111	0.793976	0.085221

Figure 23 - Score values for test set = 8 games, n_features_selected = 15

4.2.3 FNN

For a detailed description of the FNN in place can be found in Section 3.5.1 FNN – Feedforward Neural Network, as well as in the code itself [5]. For the deep learning models the feature selector is not applied, instead all 109 features are fed as an input. It is assumed that the deep learning models work best if they have the full range of features available. The test set size is always consisting of the last 20 games.

Due to the time computational time consumption of the model, we focus on one single Dataframe, the one from the team Sion only, in this Section. Following the different outcome of the model (shown in form of loss function, see Figure 24), based on the different Hyperparameter settings (see Table 2). Those results are discussed in more details in Section 4.4 Discussion.

Hyperparameter	Characteristic
n_features	109
Test set size	20 games
Batch size	10
epochs	250
alpha	0.005
Hidden layers	3 / af = 'linear', 'relu', 'relu' / units: 109,30,15

Table 2 - Hyperparameter setting 1, FNN

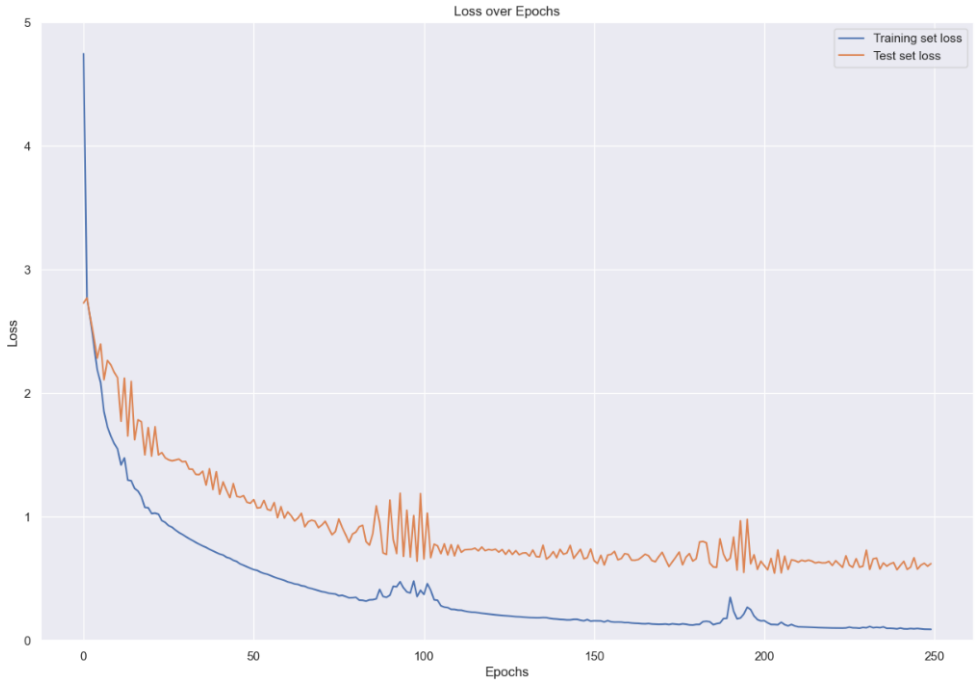

Figure 24 - Comparison of loss function from test and training set, Hyperparameter setting 1, FNN

The predicted goals on the test set, plotted against the actual goals, coming from this model, are visualized in Figure 25.

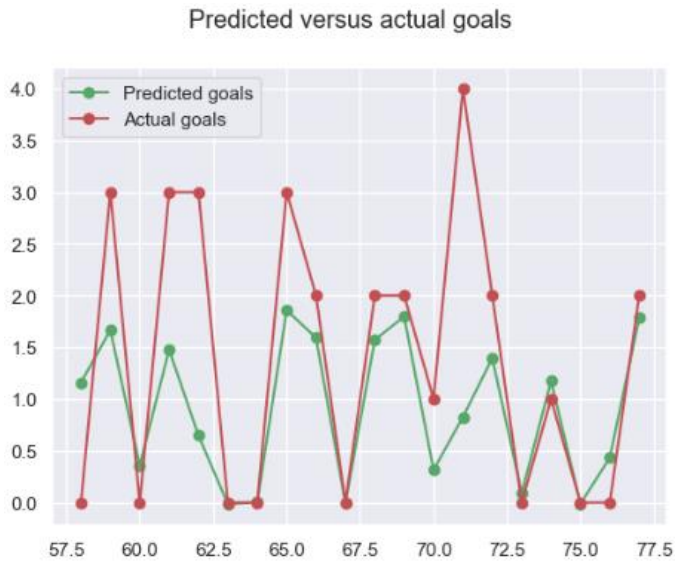

Figure 25 - Predicted vs. actual goals, Hyperparameter setting 1, FNN

In the 2nd setting, the number of hidden layers are reduced to 2, see Table 3. In Figure 26, the loss function for the test and training set, calculated by this model, are visualized.

Hyperparameter	Characteristic
n_features	109
Test set size	20 games
Batch size	10
epochs	250
alpha	0.005
Hidden layers	2 / af = 'linear', 'relu' / units: 109,30

Table 3 - Hyperparameter setting 2, FNN

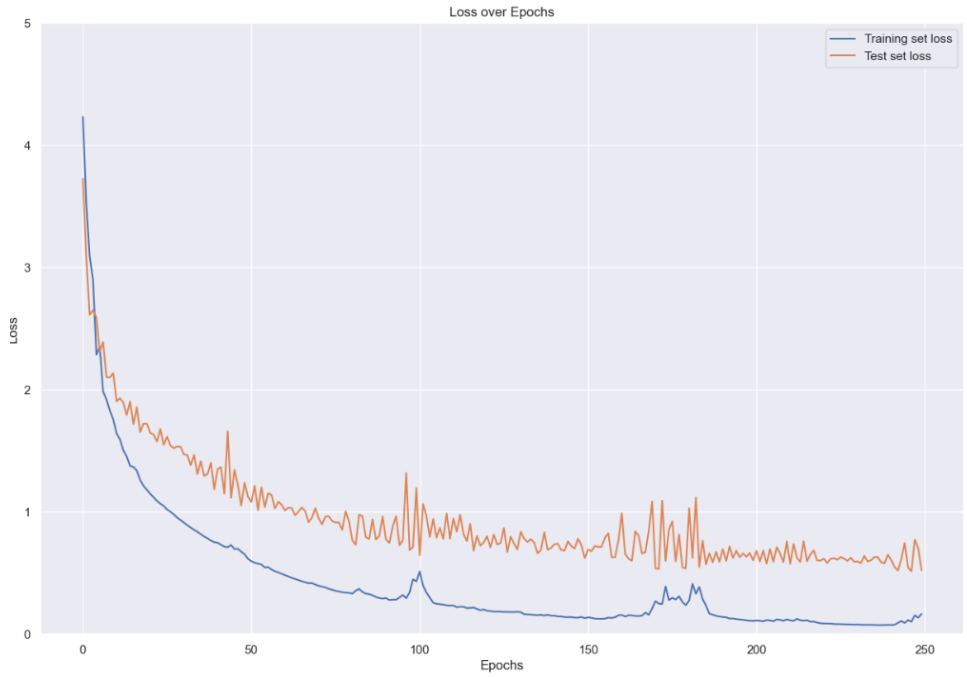

Figure 26 - Comparison of loss function from test and training set, Hyperparameter setting 2, FNN

The predicted goals on the test set, plotted against the actual goals, coming from this model, are visualized in Figure 27.

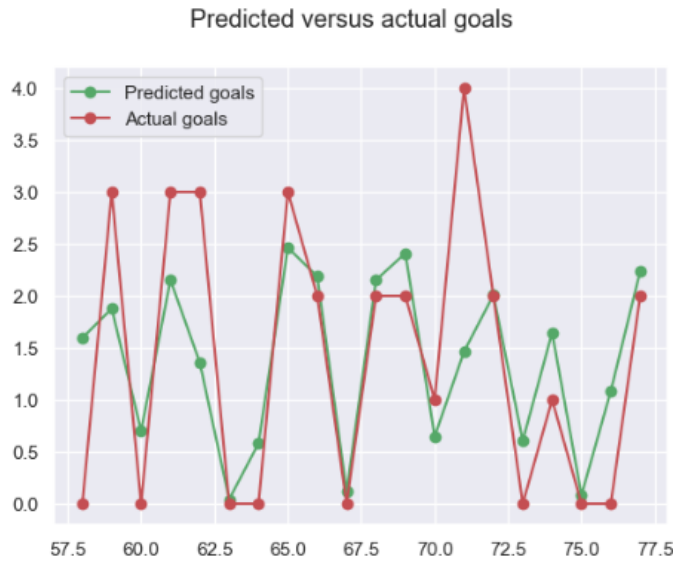

Figure 27 - Predicted vs. actual goals, Hyperparameter setting 2, FNN

In the 3rd setting for the FNN, the number of hidden layers are still set to 2 but *alpha* value is reduced to 0.001 and *epochs* are increased to 300, see Table 4. In Figure 26, the loss function for the test and training set, calculated by this model, are visualized.

Hyperparameter	Characteristic
n_features	109
Test set size	20 games
Batch size	10
epochs	300
alpha	0.001
Hidden layers	2 / af = 'linear', 'relu' / units: 109,30

Table 4 - Hyperparameter setting 3, FNN

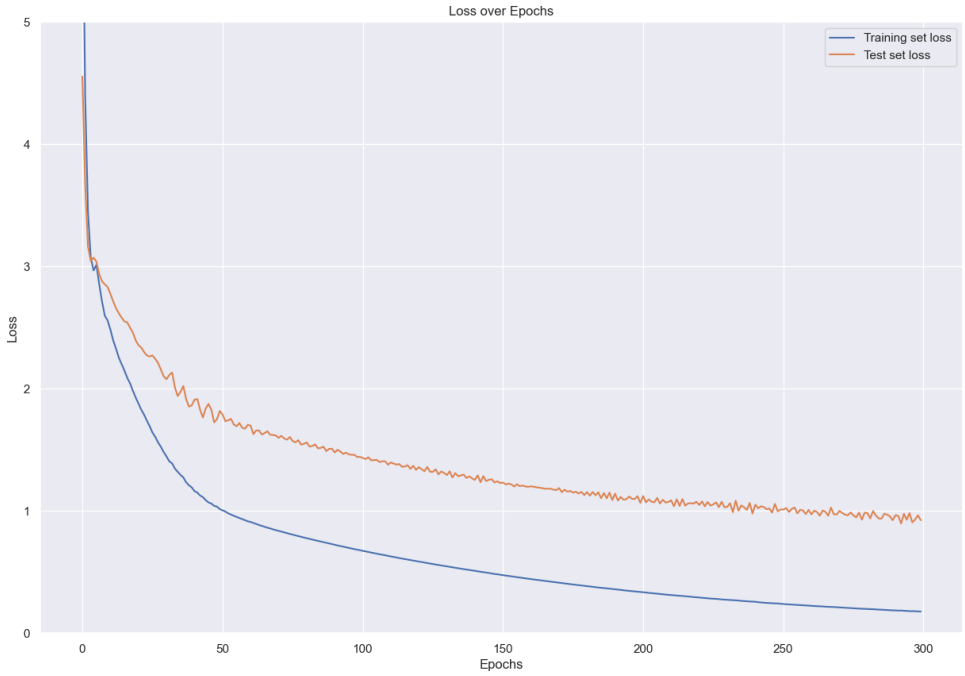

Figure 28 - Comparison of loss function from test and training set, Hyperparameter setting 3, FNN

The predicted goals on the test set, plotted against the actual goals, coming from this model, are visualized in Figure 29.

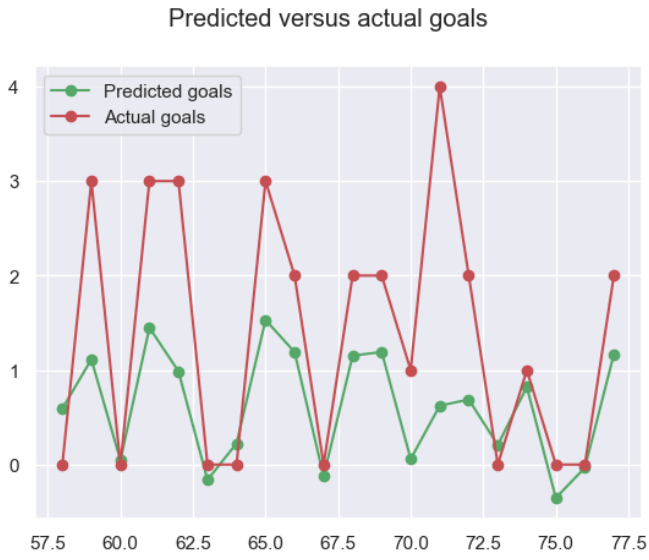

Figure 29 - Predicted vs. actual goals, Hyperparameter setting 3, FNN

One can say model 2 with two hidden layers (see Table 3), based on the loss function of the test set, works best for this kind of data.

4.2.4 RNN

In order to compare the output of a RNN with the one of the FNN, also in this section, as an example and to be consistent, the data set of Sion is used only. As mentioned in the previous Section, 109 features work as an input and no feature selector is applied. For more details about the RNN see Section 3.5.2 RNN – Recurrent Neural Network and the code [5].

The set hyperparameter settings are listed in Table 5.

Hyperparameter	Characteristic
n_features	109
Test set size	20 games
Batch size	5
epochs	150
Alpha	0.005
Hidden layers	3 / af = 'relu', 'relu', 'relu' / units: 109,30,15

Table 5 - Hyperparameter setting 1, RNN

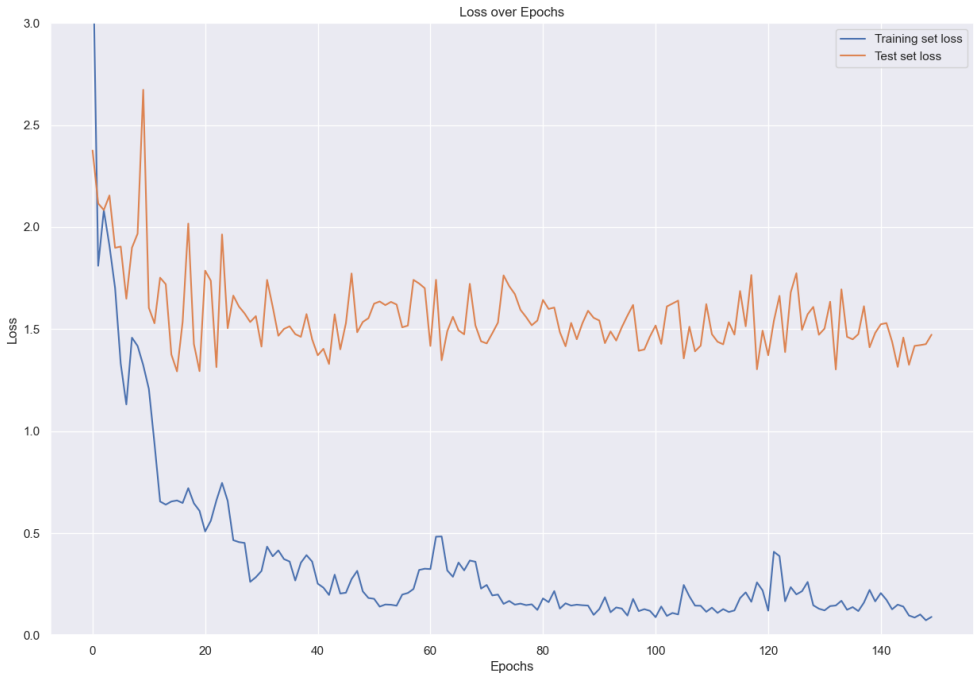

Figure 30 - Comparison of loss function from test and training set, Hyperparameter setting 1, RNN

The predicted goals on the test set, plotted against the actual goals, coming from this model, are visualized in Figure 31.

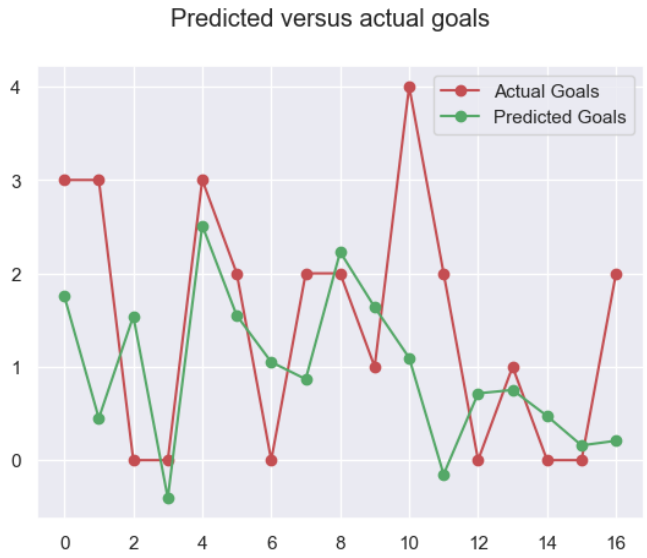

Figure 31 - Predicted vs. actual goals, Hyperparameter setting 1, RNN

After further adjusting the hyperparameter, it shows that the settings of this section are close to the optimum. However, based on the loss function value of the test set and the noisy graph, it looks like the RNN is not really working with the data set on hand.

4.3 Valuation of predicted goals

In this section, the predicted number of goals based on the different models and their *score* values from Section 4.2 Valuation of applied Machine learning models are listed. For the approach on how those goals are predicted, please refer to Section 3.6 Predicting the number of goals. As the *score* values described in Section 4.2.2 Setting 2 with a test set size of 20 games and 15 features selected looks the most promising, the random forest regression and the LASSO models are used to predict the goals of the future games.

The simulated games and their actual outcome (round 17 of the season 2022/2023) are listed in Table 6.

Team_h	Team_a	Goals_h	Goals_a
LUZ	ZUR	2	2
GRA	YB	1	2
SIO	LUG	2	3
SG	BAS	1	1

Table 6 - Game that are predicted and the actual outcome

Following different calculations, based on how many games of the past are considered to predict the next game, are listed.

Games considered from the past to build a new data record on the test set: 2

Test set size: 20 latest games

The outcome of this setting is shown in Figure 32.

	Team_h	Team_a	H_score_RF	A_score_RF	H_score_Lasso	A_score_Lasso
0	LUZ	ZUR	1.560000	2.133333	1.467368	2.144218
1	GRA	YB	2.200000	2.760000	1.996920	2.354619
2	SIO	LUG	1.173333	1.280000	0.497135	1.175749
3	SG	BAS	1.666667	1.440000	2.246491	1.544389

Figure 32 - Predicted goals, considering the last two home or away games to build a new record on the test data

The probabilities shown in Figure 33 are based on the predicted goals from the random forest regression model. Columns starting with 'Odd*', are calculated odds, as you find them on betting portals. This number are used in case one want to compare the probabilities with the bet odds. Bet odds are calculated by 1/probabilitiy. E.g. for the team LUZ the probability to win is 0.28. Then, the according bet odd is 1/0.28 = 3.54.

	Team_h	Team_a	Win	Draw	Loss	Both_score	Plus_0.5	Plus_1.5	Plus_2.5	Plus_3.5	Odd_W	Odd_D	Odd_L	Odd_both_score	Odd_Plus_0.5	Odd_Plus_1.5	Odd_Plus_2.5	Odd_Plus_3.5
0	LUZ	ZUR	0.282441	0.208193	0.480943	0.696311	0.975111	0.883188	0.713437	0.504454	3.540560	4.803233	2.079248	1.436140	1.025524	1.132262	1.401666	1.982340
1	GRA	YB	0.313527	0.179187	0.449857	0.832918	0.992987	0.958203	0.871938	0.729314	3.189519	5.580769	2.222927	1.200599	1.007063	1.043620	1.146870	1.371151
2	SIO	LUG	0.338096	0.272576	0.425288	0.498635	0.913994	0.702991	0.444162	0.232496	2.957737	3.668699	2.351348	2.005475	1.094100	1.422493	2.251433	4.301142
3	SG	BAS	0.430667	0.236616	0.332717	0.618947	0.955250	0.816227	0.600278	0.376651	2.321978	4.226262	3.005557	1.615648	1.046846	1.225149	1.665894	2.654977

Figure 33 - Probabilities for different outcome of a game, based on the predicted number of goals, setting 1

Games to considered from the past to build a new data record on the test set: 4

Test set size: 20 latest games

The outcome of this setting is shown in Figure 34.

	Team_h	Team_a	H_score_RF	A_score_RF	H_score_Lasso	A_score_Lasso
0	LUZ	ZUR	1.533333	1.786667	1.590397	1.302052
1	GRA	YB	2.000000	3.466667	0.738293	2.290849
2	SIO	LUG	1.080000	1.253333	0.888695	1.302775
3	SG	BAS	1.560000	1.360000	1.893962	1.548850

Figure 34 - Predicted goals, considering the last four home or away games to build a new record on the test data

The probabilities shown in Figure 35 are based on the predicted goals from the random forest regression model.

	Team_h	Team_a	Win	Draw	Loss	Both_score	Plus_0.5	Plus_1.5	Plus_2.5	Plus_3.5	Odd_W	Odd_D	Odd_L	Odd_both_score	Odd_Plus_0.5	Odd_Plus_1.5	Odd_Plus_2.5	Odd_Plus_3.5
0	LUZ	ZUR	0.333085	0.227631	0.421489	0.652820	0.963847	0.843820	0.644574	0.424076	3.002237	4.393077	2.372540	1.531816	1.037509	1.185087	1.551412	2.358068
1	GRA	YB	0.195306	0.146119	0.559268	0.837666	0.995772	0.972676	0.909541	0.794495	5.120166	6.843752	1.788051	1.193793	1.004246	1.028091	1.099455	1.258662
2	SIO	LUG	0.317540	0.279930	0.437034	0.471825	0.903028	0.676760	0.412781	0.207464	3.149207	3.572327	2.288152	2.119430	1.107385	1.477628	2.422593	4.820121
3	SG	BAS	0.421733	0.245426	0.332841	0.587137	0.946066	0.788580	0.558650	0.334851	2.371168	4.074551	3.004436	1.703180	1.057008	1.268102	1.790030	2.986401

Figure 35 - Probabilities for different outcome of a game, based on the predicted number of goals, setting 2

Games to considered from the past to build a new data record on the test set: 6

Test set size: 20 latest games

The outcome of this setting is shown in Figure 36.

	Team_h	Team_a	H_score_RF	A_score_RF	H_score_Lasso	A_score_Lasso
0	LUZ	ZUR	2.106667	1.693333	2.123312	1.390004
1	GRA	YB	1.786667	3.413333	0.591781	2.285627
2	SIO	LUG	1.080000	1.253333	1.054738	1.202972
3	SG	BAS	1.800000	1.426667	1.972135	2.153705

Figure 36 - Predicted goals, considering the last six home or away games to build a new record on the test data

The probabilities shown in Figure 37 are based on the predicted goals from the random forest regression model.

Team_h	Team_a	Win	Draw	Loss	Both_score	Plus_0.5	Plus_1.5	Plus_2.5	Plus_3.5	Odd_W	Odd_D	Odd_L	Odd_both_score	Odd_Plus_0.5	Odd_Plus_1.5	Odd_Plus_2.5	Odd_Plus_3.5
LUZ	ZUR	0.47675	0.208836	0.294205	0.716823	0.977629	0.892620	0.731103	0.526515	2.097536	4.788449	3.398991	1.395045	1.022883	1.120297	1.367796	1.899281
GRA	YB	0.16966	0.142310	0.601295	0.805065	0.994481	0.965797	0.891213	0.761935	5.894156	7.026931	1.663076	1.242135	1.005550	1.035414	1.122066	1.312449
SIO	LUG	0.31754	0.279930	0.453415	0.471825	0.903028	0.676760	0.412781	0.207464	3.149207	3.572327	2.205487	2.119430	1.107385	1.477628	2.422593	4.820121
SG	BAS	0.46452	0.229045	0.306435	0.634283	0.960310	0.832245	0.625634	0.403412	2.152759	4.365952	3.263339	1.576584	1.041330	1.201569	1.598379	2.478858

Figure 37 - Probabilities for different outcome of a game, based on the predicted number of goals, setting 3

4.4 Discussion

In average, the random forest regression model delivers the best *score* values. However, it is interesting to see that although the LASSO model has lower *score* values in the training set, it performs better on the test set than the random forest regression model. It seems, therefore, that the random forest regression model over-fits the training data or at least more than the LASSO model, as the *score* values on the training set are constantly above 93%. Thus, the LASSO model appears to handle test sets that differ significantly from the training set better. This means that test sets which have a poor *score* value calculated by the random forest regression model, are better handled by the LASSO model (see e.g., Figure 18). Generally, the KNN model does not seem to work for the problem / data at hand.

Obviously, the size of the test set has an impact on the *score* value of the models. It was found that the bigger the test data size, the better the outcome of the *score* value. This is especially visible in test sets that have apparently more noise, compare, for example, the result of the *score* value for Lugano when the test set comprises the last 12 games as opposed to when it comprises the last 20 games (Figure 16 & Figure 20).

The number of data records per team has an impact on the performance, i.e. the accuracy, of the model as well. This can be seen in the figures of Section 4.2 Valuation of applied Machine learning models for the team 'GRA', for example in Figure 20. While for the random forest regression model all other teams have a *score* value above 96% on the training set, 'GRA' is the only team to have a lower value. This is consistent for all settings elaborated, and can be explained with the fact that 'GRA' is the team with the smallest data set size (51 instead of 88 data records). This is because there is data missing for 'GRA' for the season 2020 / 2021 due to the fact that the team promoted in that season from the 2nd league to the Super

League. Although there would be available data from that season for the team, it was not considered here, as it is assumed that the performance is not representative in another league against other teams. This shows that the number of data records at least has an impact on the *score* value of the training model. However, this does still not mean that the *score* value must be low on the test set as well. 'GRA' does not drastically underperform on the test set compared to other teams.

As shown in Section 4.2Valuation of applied Machine learning models, the deep learning models seem to work less well than the machine learning models, whereas the FNN shows better results than the RNN. Obviously, the number of available records appear to affect those models. The models look like they work well on the training data but fail to work good enough on the test data. In order to improve their performance, the test data size has to be enhanced. Additionally, the computational time of the deep learning models is way higher (~ 4 min for the deep learning models vs. max. 4 sec for the machine learning models, not considering the feature selector for the machine learning models).

As shown in Section 4.3Valuation of predicted goals, the number of goals in an upcoming game is sometimes better predicted by a 'worse' model. This means that although the *score* value of the LASSO model on the test data is lower than the one from the random forest model, the predicted number of goals per team calculated by the LASSO model are closer to the actual result. This can be explained by the complexity of the problem statement and the uncertainties. One would have to compare more than just one game to make conclusions about that fact.

Regarding the optimal number of past games to consider in order to build a new data record of the test set and to predict the goals of a future game with it, one has to simulate more than just one game as well. However, for the games analyzed in this project, the best number of past games to consider seems to be two.

5. Conclusion

After answering the objectives of this project and explaining the approaches of how this was done, there remain some uncertainties: How does one handle the start of a new season? Does it make sense to create a test set that involves a game of the past season and games of the new season? Obviously, a mathematical model does not care about this fact but it still needs to be elaborated. There have been many cases were a team performed very well in one season and failed in the next. A machine learning model potentially needs time (and data) to adjust this fact.

Whether to us a model or not in the future will be dependent on a threshold of the *score* value that has yet to be defined. That is, the predicted numbers of goals are only accurate and therefore useful when the *score*

value of the test set is above this threshold. Additionally, one has to consider whether to concede with only one model (random forest regression) or to continue with two, preferably adding the LASSO model. Like this, one can choose the best *score* value of the test set and combine the model in that sense when predicting goals of future games.

It also has to be checked whether it makes sense to apply the whole data set for the deep learning approaches. That is, instead of creating a model for each team, one general model is built. Like this, a RNN, for example, would have more data available on the test set and potentially would work better or even exceed the performance of the random forest regression model.

Questionable is also whether the regression approach is the most efficient. In the literature, game outcomes are mainly described as classification problems. But I argue that the output should be the same in the end and with the approach of a regression problem there are even more possibilities to define different outcomes, such as the number of scored goals.

Also, in terms of available features, there is room to improve. For example, data about players are not yet included at all. One expects to further improve the model by adding such information. Simultaneously, the needed (weekly) data extract has to be automatized in order to run the code efficiently. As explained, this is currently not yet the case.

All in all, the models work good enough to make useful predictions and to compare them, for example against betting odds. By fine tuning the code, the whole case can become economical.

Bibliography

[1] V. S. Olli Heino, "Forecasting football match results - A study on modeling," 2013.

[2] S. M. I. Yoel F. Alfredo, "Football Match Prediction with Tree Based Model Classification," *I.J. Intelligent Systems and Applications,* 2019.

[3] P. Pechta, "https://github.com/pawelp0499/football-prediction-model," [Online].

[4] "InStat Football," https://football.instatscout.com/, [Online]. Available: https://football.instatscout.com/.

[5] N. Gfeller, "github," 2023. [Online]. Available: https://github.com/BigHarryG/CAS-ADS/tree/main/Final%20Project.

[6] S. Mishra, 2020. [Online]. Available: https://towardsdatascience.com/methods-for-normality-test-with-application-in-python-bb91b49ed0f5.

[7] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/D%27Agostino%27s_K-squared_test.

[8] "pandas documentation," NumFOCUS, Inc., 2023. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html.

[9] "Pandas," NumFOCUS, Inc., 2023. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html.

[10] "scikit-learn," 2007 - 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[11] "Pandas," NumFOCUS, Inc, 2023. [Online]. Available: https://pandas.pydata.org/docs/index.html.

[12] s.-l. developers, "scikit-learn," 2007-2023. [Online]. Available: https://scikit-learn.org/stable/getting_started.html.

[13] T. M. d. team, "Matplotlib," 2012-2023. [Online]. Available: https://matplotlib.org/.

[14] N. Developers, "NumPy," 2008-2022. [Online]. Available: https://numpy.org/doc/stable/.

[15] T. S. community, "SciPy," 2008-2023. [Online]. Available: https://docs.scipy.org/doc/scipy/.

[16] "TensorFlow," [Online]. Available: https://www.tensorflow.org/overview.

[17] "scikit-learn," scikit-learn developers, 2007 - 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html.

[18] G. Chauhan, "Machine Learning HD," 14 March 2021. [Online]. Available: https://machinelearninghd.com/ridgecv-regression-python/.

[19] s.-l. developers, "scikit-learn," 2007 - 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html.

[20] s.-l. developers, "scikit-learn," 2007 - 2023. [Online]. Available: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.

[21] s.-l. developers, "cikit-learn," 2007 - 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[22] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *springer,* p. 21, 2021.

[23] s.-l. developers, "scikit-learn," 2007 - 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html.

[24] I. Muhajir, "medium," 2019. [Online]. Available: https://medium.com/analytics-vidhya/k-neighbors-regression-analysis-in-python-61532d56d8e4.

[25] B. T., "towardsdatascience," 2021. [Online]. Available: https://towardsdatascience.com/intro-to-scikit-learns-k-nearest-neighbors-classifier-and-regressor-4228d8d1cba6.

[26] H. Onnen, "towardsdatascience," 31 October 2021. [Online]. Available: https://towardsdatascience.com/temporal-loops-intro-to-recurrent-neural-networks-for-time-series-forecasting-in-python-b0398963dc1f.

[27] Z. P. S. C. K. e. a. Che, "Recurrent Neural Networks for Multivariate Time Series with Missing Values," *Sci Rep,* 2018.

[28] P. Sanagapati, "kaggle," 2019. [Online]. Available: https://www.kaggle.com/code/pavansanagapati/what-is-dropout-regularization-find-out.

[29] k. team, "Keras," [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/.

[30] "stackoverflow," 2021. [Online]. Available: https://stackoverflow.com/questions/66952606/what-is-this-flatten-layer-doing-in-my-lstm.