Vrije Universiteit Amsterdam Forward Football

Master Thesis

Exploring Machine Learning Approaches for Football Shot Detection

A comparative study

Author: I.J. de Lange (2709091)

1st supervisor:	A.E. Eiben	
daily supervisor:	Noosheen Gholami	Forward Football
2nd reader:	R.D. van der Mei	

A thesis submitted in fulfillment of the requirements for the VU Master of Science degree in Business Analytics

July 16, 2023

Abstract

This thesis explores the application of machine learning techniques to automate the detection of shots on goal in football matches using local positioning measurement (LPM) systems. By leveraging object-tracking data and extracting relevant features, machine learning algorithms such as random forest, XGBoost, neural network, and recurrent neural network were utilized. Different sampling techniques, including SMOTE, borderline SMOTE, undersampling, and no sampling, were combined with these methods to optimize shot detection accuracy while minimizing false positives. Evaluation metrics such as precision, recall, F1 score, and AUC-ROC were employed to compare the performance of various method combinations. A tolerance interval of 2 seconds was allowed to detect shots. The results showed that tree ensembles (random forest and XGBoost) achieved the best performance with borderline SMOTE sampling, while neural network models performed well without any sampling. Undersampling yielded mixed results, except for the neural network, where it surprisingly outperformed other approaches. Challenges in noise detection led to higher false positives, with misclassification of passes near or towards the goal as shots being a common issue. The direction and distance of the ball from the goal were identified as crucial features for shot detection. Overall, machine learning can effectively detect football shots, and the combination of neural network and undersampling demonstrated the highest F1 score.

Executive summary

Context. Football is a popular sport worldwide, and analyzing game events has always been important. Traditionally, events like shots on goal were manually annotated based on visual observation. However, the emergence of analysis software and machine learning has led to the development of automated systems that provide precise player and ball coordinates during matches. These systems, known as Local Positioning Measurement (LPM) systems, allow for the collection of tracking data multiple times per second. This data enables machine learning algorithms to automatically detect game events, eliminating the need for manual coding.

Goal. This thesis addresses the challenges in manually collecting event data for football by investigating the application of machine learning techniques. Its primary focus is on automatically detecting shots on goal using object-tracking data. The research question, *To what extent can machine learning be employed for the detection of football shots?* aims to determine the feasibility of using machine learning for precise shot detection, aiming to optimize accurate detections while minimizing false positives.

Method. Using the raw positional data, features were extracted to capture the ball trajectory for shot detection. Various machine learning and sampling techniques, including random forest, XGBoost, neural network, and recurrent neural network, were employed. These methods were combined with different sampling techniques such as SMOTE, borderline SMOTE, undersampling, and no sampling. A thorough evaluation compared the performance of different method combinations using precision, recall, F1 score, and AUC-ROC as evaluation metrics. To improve model performance, a tolerance interval of 2 seconds was allowed when detecting shots.

Results & conclusions. After evaluation, it was seen that the tree ensembles (random forest and XGBoost) achieved the best performance with the borderline SMOTE sampling technique, while the neural network and recurrent neural network performed well without any sampling. Undersampling yielded mixed results, performing poorly on all models, except for the neural network, where it surprisingly outperformed other approaches. All models faced challenges in noise detection, resulting in more false positives than true positives. The models frequently misclassified passes near or towards the goal as shots. The direction of the ball towards the goal and its distance from the goal were identified as the most important features for shot detection. Overall, machine learning can be effectively utilized for football shot detection, with the combination of the neural network and undersampling yielding the highest F1 score.

Acknowledgements

This thesis represents the final piece of my master's in Business Analytics at the Vrije Universiteit Amsterdam. The research focused on detecting football shots using player and ball coordinates and was conducted during a 20-week internship at the data science department of Forward Football.

Throughout this internship, I received exceptional guidance from my internal supervisor, Noosheen Gholami. Her willingness to challenge my ideas and provide valuable feedback greatly enriched my work. I am sincerely grateful for her support.

I would also like to extend my gratitude to my university supervisor, Guszti Eiben, for his excellent guidance, mentorship, and flexibility. The combined efforts of my supervisors ensured a smooth graduation process, for which I am truly thankful. I would also like to express my appreciation to Rob van der Mei for his role as the second reader of my thesis.

My time at Forward Football has been a transformative experience, contributing to both my personal and professional growth. It has significantly expanded my knowledge and skills in the field of data science, as well as extending my understanding of football. I am eager to apply these newfound abilities in my future endeavours.

Isis de Lange

June 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Football is a globally popular sport with millions of fans and players worldwide. The analysis of football matches has always intrigued teams, media, experts, and fans, aiming to gain deeper insights into on-pitch activities. Key game events such as passing, interceptions, and, most importantly, shots on goal have been of primary interest. Traditionally, such events were manually annotated based on qualitative analysis of broadcast footage, heavily reliant on visual observation.

However, the advent of analysis software and machine learning has opened up possibilities for automating this process. Consequently, there has been a significant increase in demand for tracking data in football. Specifically, there is interest in technologies that can provide precise coordinates of players and the ball multiple times per second. In response, Local Positioning Measurement (LPM) systems have emerged in the market, enabling the collection of player and ball coordinates during matches. These systems have evolved over time, offering opportunities to quantify previously qualitative observations and facilitate event detection.

The rise of LPM systems has paved the way for event identification using tracking data, eliminating the need for manual coding. As a result, there is a growing interest in developing machine learning algorithms capable of automatically detecting game events using tracking data. This advancement holds immense potential to enhance the accuracy and efficiency of event detection.

1.2 Problem statement

The manual collection of event data for different football leagues by commercial providers presents several challenges. Firstly, there is a lack of consistency in event definition, terminology, granularity, and accuracy of time annotation across different providers due to the subjective nature of event detection. Additionally, the manual annotation process itself is time-intensive and prone to errors.

To address these challenges, this thesis explores the potential of utilizing machine learning techniques to automatically detect shots on goal in football games using object-tracking data as input. The central research question of this thesis is:

$$\boxed{\text{To what extent can machine learning be employed for the detection of football shots?}}$$

The main goal of the research is to successfully detect as many shots as possible while minimizing the number of incorrectly detected shots.

1.3 Host organization

This research project was conducted through a collaborative effort between the Vrije Universiteit van Amsterdam and Forward Football. Forward Football is a start-up that leverages technology, data analysis, and academic insights to enhance football practices for clubs and coaches. One of their core services is pass-performance analysis, which assesses a player's success rate in passing. Currently, the events analyzed by Forward Football are detected and provided by an undisclosed third party, which may not consistently guarantee the desired reliability.

By developing an accurate and efficient machine learning-based system for detecting shots on goal in football games, this research aims to significantly enhance Forward Football's ability to deliver reliable and effective analysis to their clients. The successful implementation of such a system will enable Forward Football to continue providing high-quality services, contributing to the ongoing improvement of football practices.

1.4 Thesis outline

The rest of this paper will be structured as follows: Chapter 2 reviews the existing literature on detection events in football using machine learning techniques. Chapter 3 explores the data used for the research, including feature engineering and pre-processing. Chapter 4 explains the machine learning methodologies employed for shot detection, along with methods to combat class imbalance and evaluation metrics. Chapter 5 presents the results obtained from the application of the developed techniques. Chapter 6 discusses the limitations of the project and proposes potential directions for future research. Chapter 7 provides a conclusion summarizing the main findings of the research.

Chapter 2

Literature review

This chapter provides an in-depth analysis of existing research on shot detection in football, specifically focusing on the utilization of machine learning techniques. The review examines various approaches employed in shot detection, encompassing both rule-based methods and machine learning-based methods. Additionally, the review identifies gaps in the current research landscape.

Sports provide a wealth of information in various forms, including statistics and live video feeds. However, the data from each source may not be compatible, which can make it challenging to obtain a comprehensive understanding of what is happening.

Manual football event annotation has been conducted since as early as 1968 (1), and in 1986, Franks and Miller (2) demonstrated the unreliability of human observation. Their experiments revealed that the expert observer's recollection of major game events could be as low as 42%. Therefore, there is a growing need for automatic event detection in order to address these limitations.

Automatic video detection Automatic video detection has gained significant attention in recent years for capturing and analyzing sports gameplay. However, a survey conducted by Gudmundsson and Horton (3) in early 2016 revealed that only a limited number of attempts had been made to automatically process ball-related events. Most of the existing research in this field has focused on detecting events directly from video recordings of matches, leveraging advancements in deep learning techniques.

For instance, Khan et al. (4) proposed a framework in 2018 that directly detects events from video streams. Their approach involves detecting objects such as the ball and players in each video frame, generating a set of candidate objects. Event detection is then performed based on temporal and logical combinations of these detected objects using predefined rules. The system achieved accuracy ranging from 84% to 92% for different events. However, it was unable to detect shots on target due to the complexity of the event, which required identifying the goalpost in addition to the ball and players.

In another study, Jiang et al. (5) employed deep learning techniques to develop an automated event detection system specifically for detecting corners, goals, and goal attempts. Their system utilized convolutional neural networks to extract relevant features from video frames and classify them accordingly. Although the model achieved high precision and recall scores, it is important to note that the detection of shots is highly subjective, leading to variations in performance depending on individual definitions and judgments. This subjective nature might limit the model's robustness for users with different criteria or interpretations.

Predicting future events Prior to 2016, a number of studies were dedicated to the prediction of future events in sports. One such study by Wei et al. (6) aimed to forecast complex sports events, such as the future position of the ball during a football match. To achieve this, the researchers utilized advanced models such as the Hidden Conditional Random Field (HCRF).

Automatic event labelling One notable attempt at automatic event labelling based on positional data, with the aid of machine learning, was conducted by Richly et al. (7, 8). In their research, they employed various machine learning methods to recognize four types of events (pass, reception, clearance, and shot on goal) in a spatiotemporal dataset. The tested methods included Support Vector Machine, K-Nearest Neighbors, Random Forest, and Artificial Neural Networks. The algorithmic results were compared to a 'golden standard' manual markup, consisting of 194 events within 8 minutes and 47 seconds of active game time. Among the tested models, Neural Networks demonstrated the best performance, achieving a precision of 89% and a recall of 90%. Notably, a smoothing filter was applied, reducing the original 25 Hz sampling rate of game recordings to 10 Hz, resulting in significant quality improvement. However, the event markup obtained did not provide extended event information, such as the identities of the passer and receiver in passing events. Additionally, the training set was relatively small, including only seven shots on target, indicating the need for a more comprehensive evaluation to assess the proposed approach's effectiveness in practical scenarios.

Another similar research study was conducted by Schuldhaus et al.(9) in 2021. Recognizing that video-based detection was predominantly available for elite teams with higher budgets, they aimed to develop a low-cost sensor-based approach for shot/pass classification in football. By utilizing accelerometer data from players' left and right shoes, they identified intensity peaks and classified them as shots, passes, or other events using segmented windows. The final system achieved an overall mean classification rate of 84.2%. Similarly, Tovinkere et al.(10) and Vidal et al. (11) presented a set of heuristic rules for detecting football events based on player and ball positions as input.

Chapter 3

Data

This chapter aims to provide a comprehensive analysis of the data used in this study, which was obtained from a third party and provided to Forward Football. The data will be described in detail, and the necessary steps taken before shot detection can be performed will be outlined. Additionally, a data exploration will be conducted to gain a deeper understanding of the dataset's characteristics and properties.

3.1 Data description

The data used in this study was collected using a Local Positioning Measurement (LPM) system and a smart ball. The LPM system comprises small trackers placed on armbands worn by all players, along with four poles strategically positioned at the corners of the field to outline the pitch. This system tracks the positions of the players and the ball, representing them as dots on a coordinate system. One of the poles serves as the origin for the coordinate system (see Figure 3.1). Following the matches, a third party performs post-processing on the data, resulting in two separate datasets containing the match information. Forward Football has access to data from a total of 35 matches.

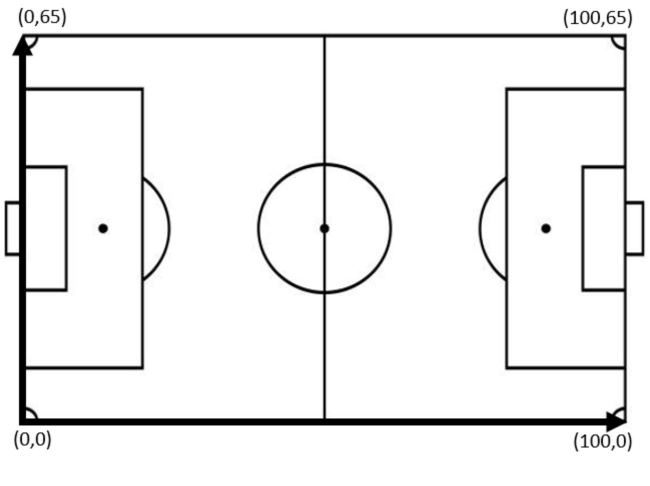


Figure 3.1: Football pitch according to LPM system.

Positional data Throughout the match, the LPM system tracks the movement of the ball and players on the pitch, generating a substantial volume of data. This data is captured at a frequency of 5Hz, resulting in five measurements per second. The resulting dataset contains meter-accurate positional data for each player and the ball throughout the entire match. It is important to note that player positions are recorded only when they are within the boundaries of the pitch area defined by the LPM system.

An illustration of the data representation with n players and k timesteps is shown in 3.1:

$$\mathrm{Pos}_{p,t} = \begin{bmatrix} x_{b,t=1} & y_{b,t=1} & x_{p_1,t=1} & y_{p_1,t=1} & \cdots & y_{p_n,t=1} \\ x_{b,t=2} & y_{b,t=2} & x_{p_1,t=2} & y_{p_1,t=2} & \cdots & y_{p_n,t=2} \\ \ldots & \ldots & \ldots & \ldots & \ddots & \ldots \\ x_{b,t=k} & y_{b,t=k} & x_{p_1,t=k} & y_{p_1,t=k} & \cdots & y_{p_n,t=k} \end{bmatrix} \tag{3.1}$$

In 3.1, $x_{b,t=1}$ represents the x-coordinate of the ball at the first timestamp, $x_{b,t=2}$ represents the x-coordinate of the ball at the second timestamp, and so on. Similarly, $x_{p_i,t=1}$ represents the x-coordinate of the i-th player in the dataset at timestamp 1.

Since the data is primarily collected during matches of young professional clubs, the typical match duration is around 60 minutes. The number of data points collected during a match highly depends on the duration and when the LPM system is activated. As the LPM system is operated by humans and is therefore susceptible to human error, it may not be activated until a few minutes after the match has begun. Consequently, the number of data points per match varies but is typically around 25,000. The number of columns in the dataset also varies, depending on the number of players, including substitutes, present in the game.

Event data In addition to the positional data, the third party also provides information on four specific match events: passes, interceptions, shots on target, and clearances. However, the exact methodology for detecting these events has not been disclosed. Shots on target are labelled with a timestamp indicating when they were presumed to have occurred during the match. They also come with additional information, such as whether the shot resulted in a goal. On average, there are typically 850 to 900 events per match, with only 20 to 40 of them being shots on target. Therefore, only a very small proportion of the data (0.2%) is labelled with the target label. It is worth noting that a shot is labelled for only one-fifth of a second (or one frame) in the match, even though a complete shot from start to finish takes longer than one-fifth of a second.

3.2 Feature engineering

It can be reasonably assumed that the raw coordinate data alone is not informative for a machine learning model. Therefore, it was necessary to manually extract more informative features from the data. The key criterion for selecting these features was to capture the context surrounding the shot event. As previously mentioned, only a single frame is labelled as a shot, even though a shot likely spans a longer duration than one-fifth of a second. Hence, including the contextual information surrounding the shot is crucial. By analyzing the positional data and considering relevant literature (7, 8) as well as personal assumptions, a set of features was derived.

Distance to goal & ball direction Given that the objective of players is to score a goal, it is reasonable to expect that a shot is likely to occur in the proximity of the target. To estimate the distance between the ball and the goal, the Euclidean distance is utilized by calculating the distance between the ball coordinates and the center point of the goal. The Euclidean distance can be computed using Equation 3.2. Since a football pitch consists of two goals, the calculation considers the nearest goal to the ball.

$$d_{b,t} = \sqrt{(x_{b,t} - x_{\text{goal}})^2 + (y_{b,t} - y_{\text{goal}})^2} \tag{3.2}$$

In addition to incorporating the current distance to the goal, supplementary attributes have been introduced to describe the spatial relationship between the ball and the goal half a second prior to and after every frame. This modification aims to enhance the algorithm's ability to perceive the ball's trajectory. These additional features enable the algorithm to differentiate between the ball's motion towards or away from the goal within a short duration of time and estimate the distance of this motion from the goal. Moreover, these features help determine the direction of the ball before and after each frame.

Ball speed The ball speed is utilized as an additional feature, as it can provide insight into the force exerted by the player during a shot. The ball speed is measured in meters per second and given by equation 3.3.

$$\text{speed} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{|\Delta t|} \tag{3.3}$$

Similar to the distance and direction features, the ball speed half a second before and after each frame is taken into account to capture any potential acceleration during a shot. This allows the algorithm to consider the changes in ball speed leading up to and following the shot event.

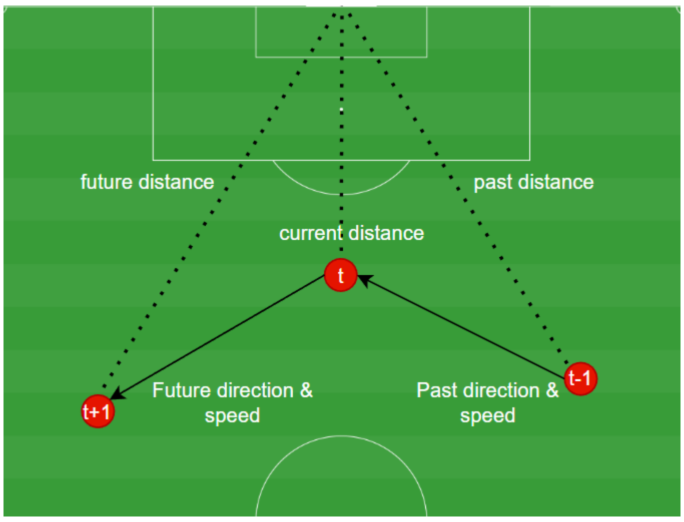


Figure 3.2: Visual outline of extracted features

A visual representation of the extracted features can be found in Figure 3.2. It is worth noting that all the extracted features are solely associated with the ball and do not encompass the players' positions, although they may contain valuable information. For example, the player's proximity to the opponent's goal could be informative. Unfortunately, this information was unavailable in the data used for this research. An effort was made to extract the ball possession information, but it was subsequently deemed unsuitable for the final dataset. The reason is that the accuracy of the ball possession output could not be verified, potentially creating confusion for any algorithms attempting to detect a shot.

3.3 Pre-processing

Before any modelling or analysis, data preprocessing is an essential step. The initial raw data used in this project was found to be clean, without any significant issues. The positional data exhibited no outliers, and the timestamps were consecutive without any gaps, except for the expected break that occurs during a football match.

Missing values Although the raw positional data did not contain any missing timestamps, missing values were still present due to the nature of sensor data. Occasional failures in data collection by player or ball sensors resulted in brief periods of missing coordinates. These missing values accounted for approximately 30.7% of all available data, with variations in the number of missing values per match. To address this issue, *linear interpolation* (12) was employed. However, it was selectively applied to address specific cases. For example, interpolation was deemed illogical for missing values occurring when a player had been substituted. Therefore, careful consideration was given to determine which missing values should undergo interpolation. Specifically, a missing value was filled only if the 30 seconds (150 frames) before and after that timestamp were also filled, indicating player or ball activity. Otherwise, the player or ball was considered inactive, and the coordinates were denoted with a dash. Following the interpolation process, the data no longer contained any missing values. It is worth noting that 26.5% of the data represented periods of player or ball inactivity, implying that the actual amount of missing data accounted for only 4.2% of the data.

One-hot encoding After conducting feature engineering, the resulting dataframe incorporated two categorical variables representing the direction of the ball before and after each frame. These variables categorized the ball direction into three options: 'towards', 'away', and 'stationary'. Since certain machine learning models cannot directly handle categorical variables, a transformation was performed using the technique of *one-hot encoding*, as described by Harris et al. (13). This process resulted in the creation of six additional columns, where each column represents one of the three categories and contains binary values indicating the presence or absence of that particular category for a given frame.

Standardization The data was normalized using the *standardization* technique, as outlined in the work by Shanker et al. (14). The primary goal of standardization in this particular context is to eliminate scale differences among features, ensuring their equal importance within the dataset. Through this process, the data is transformed to possess a mean of 0 and a standard deviation of 1, facilitating simpler comparison and analysis of different variables. Moreover, standardization aids in the convergence of machine learning algorithms, ultimately resulting in improved performance and faster training.

Removing rows To assist the algorithm in detecting shots, certain rows were removed from the dataset for two specific reasons. Firstly, these rows contained information deemed unimportant or uninformative for shot detection. Secondly, the removal of these rows addressed the issue of significant class imbalance in the data.

The middle third of the pitch was entirely removed from the dataset. Traditionally, this area is not commonly used for shooting at the goal, making it appropriate to eliminate these rows. Additionally, the ball often spends considerable time in this region, justifying the removal of a substantial portion of the data. Rows containing data where the ball was out of the pitch were also eliminated.

In addition to removing rows where the ball is out of the pitch, which may contain misclassified shot labels, it was observed that prolonged periods during a match occurred with no events. It is unlikely for a shot to happen without preceding events such as passes, interceptions, tackles, or dribbles. Therefore, if there is a lack of such events for more than 15 seconds, it is unlikely that a shot will occur, and this section is removed from the data. This ensures that only relevant and informative data is included in the analysis, leading to improved model performance.

To avoid confusion for the algorithm, rows following a shot were eliminated. The data being recurrent, maintaining information after a shot might lead to ambiguity as the data contains the same ball pattern as when the shot started. Therefore, rows were removed after a shot where the ball was still in motion towards the goal.

After feature engineering, a few outliers appeared in the dataset. These outliers resulted from sudden jumps in the positional data caused by the halftime break during a match. The sudden position jumps led to erroneous calculations for ball distance and speed. Given that it is highly unlikely for a shot event to occur within the first few seconds after the team returns to the field following the break, these rows were removed. Additionally, when the ball was shot out of the field and beyond the detection zone of the LPM system, the ball coordinates glitched, resulting in extreme speed and distance values. These outliers were also removed.

Overall, this process removed approximately 83% of the total data.

3.4 Exploration

To gain deeper insights into the data, exploratory data analysis (EDA) was conducted. EDA played a crucial role in uncovering patterns, relationships, distributions, and other valuable insights within the dataset.

The final dataset consists of approximately 142,000 frames of data collected from 35 football matches. The distribution of the target 'event' is illustrated in Figure 3.3. Due to the mature of football matches, the data exhibits a severe imbalance, with a mere 0.38% (540 frames) representing the 'shot' label.

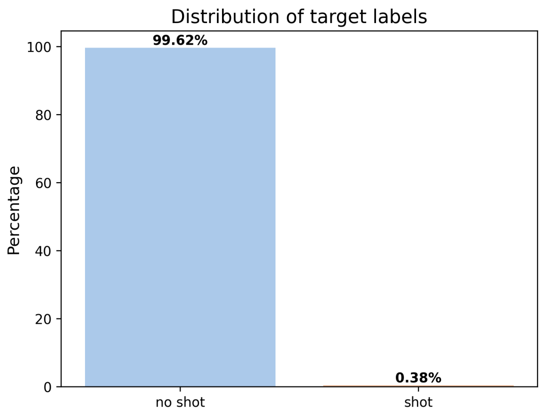
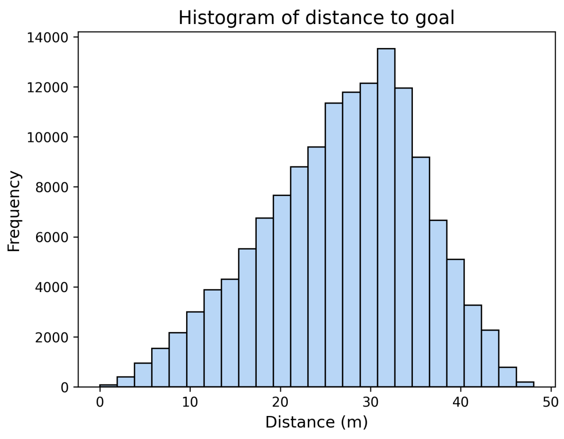
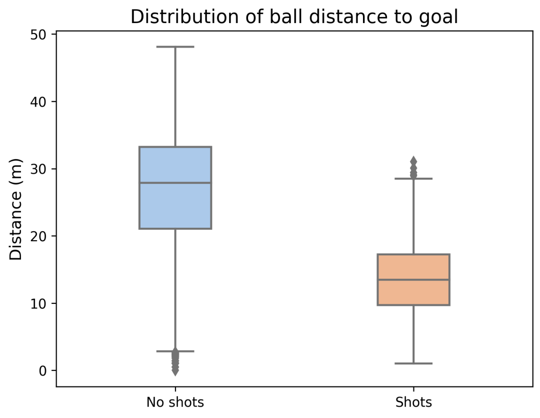


Figure 3.3: Distribution of the target label



(a) Distance to goal



(b) Distance to goal of shots

Figure 3.4: Distribution of the distance to the goal

The distance between the ball and the goal is a crucial factor in determining the appearance of a shot. Figure 3.4a displays a histogram depicting the distribution of distances, revealing that the typical range between the ball and the goal is 21 to 33 meters, with an average distance of 26.9 meters across the entire dataset. The distribution appears to be slightly left-skewed.

Examining Figure 3.4b, we observe a significant difference in the distribution of distances when a shot is detected. When a shot is detected, the distance to the goal visibly decreases, ranging from 9 to 17 meters, with an average distance of 13.8 meters. While a few outliers can be observed during shots, the distance to the goal never exceeds 31 meters. It is important to note that the distance to the closest goal is considered, as a football pitch has two goals. Therefore, when the ball crosses the middle line, it is considered closer to the opposite goal, which explains why the distance to the goal never exceeds 50 meters.

To further investigate the relationship between the ball's distance to the goal and the occurrence of a shot, Figure 3.5 presents a violin plot showcasing the distribution of the

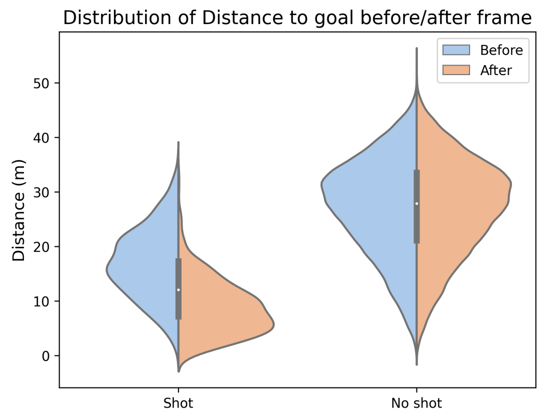


Figure 3.5: Distribution of the distance to the goal before and after each frame

ball's distance to the goal before and after each frame. The plot features two violins: one representing the distance to the goal before and after a shot, and the other representing the same for regular frames without a shot. The violins display the distance distribution in blue for before a frame and in orange for after a frame.

The plot is particularly interesting because it allows for a clear visual comparison of the distributions. Examining the violin corresponding to the shot labels, it is evident that before a shot is taken, the distance predominantly falls between 12 and 21 meters. However, after a shot is taken, the distance significantly reduces, primarily ranging between 5 and 11 meters. This indicates that the ball moves considerably closer to the goal after a shot is taken. Conversely, the plot corresponding to the other frames, representing non-shot events, demonstrates that the distribution remains largely unchanged.

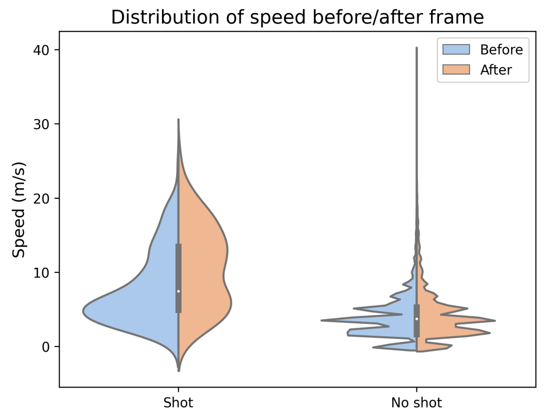
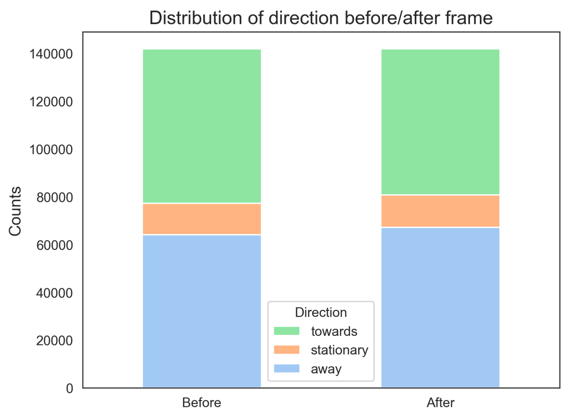


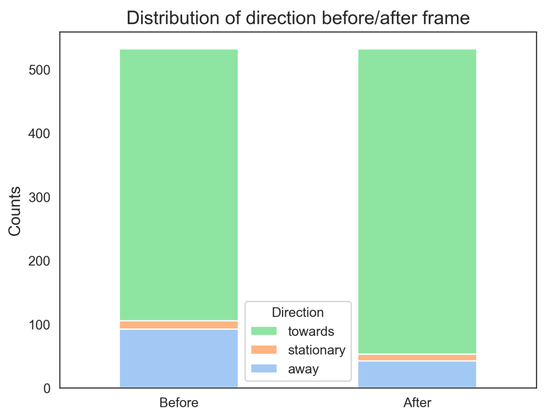
Figure 3.6: Distribution of the ball speed

To examine the relationship between ball speed and the occurrence of a shot, a similar analysis can be conducted. Figure 3.6 presents a violin plot showcasing the distribution of ball speed before and after each frame.

The plot reveals that in the absence of a shot, the distribution of ball speed before and after each frame is comparable, with most values falling between 2 and 5 meters per second. However, when a shot is detected, a noticeable difference is observed in the distribution of ball speed before and after the frame. This discrepancy can be attributed to the players exerting increased force while attempting a shot in order to bypass the goalkeeper. This observation is supported by the data, which indicates that ball speed before a shot typically ranges from 3 to 10 meters per second, while the speed after a shot ranges from 6 to 14 meters per second.



(a) Ball direction



(b) Ball direction of shots

Figure 3.7: Distribution of the direction to the goal

Finally, Figure 3.7a depicts the direction of the ball relative to the center of the goal before and after each frame. The figure demonstrates that there is no discernible pattern in the ball's direction across the entire dataset. The ball appears to move both away from and towards the center of the nearest goal in roughly equal proportions, with only a small percentage of time spent stationary.

However, upon closer examination of the direction before and after each frame during a shot, a clear distinction emerges. It becomes evident that during a shot, the direction of the ball is predominantly towards the goal. Nevertheless, it is important to note that this is not always the case, as there are instances where the ball moves away from the goal half a second before or after the shot is detected. This can occur due to various reasons, including the possibility of faulty labels.

Chapter 4

Methodology

This chapter serves as a roadmap for the techniques used to detect shots, including Random Forest, Extreme Gradient Boosting, neural networks, and recurrent neural networks. It also describes several sampling techniques used to combat class imbalance. Lastly, it outlines the evaluation process for the models.

4.1 Machine learning techniques

4.1.1 Random Forest

Random Forest, first proposed by Ho et al. in 1995 (15) and introduced by Breiman et al. in 2001 (16), is an ensemble learning method widely used for classification, regression, and various other tasks. It leverages the combination of multiple decision trees to make predictions. In this section, we provide an overview of the theoretical foundations underlying the Random Forest algorithm.

At the core of Random Forests are Decision Trees, which serve as the fundamental building blocks for the ensemble. Decision trees are hierarchical models that partition the feature space to make predictions. They consist of nodes and branches, with the root node representing the initial feature space and subsequent nodes representing feature splits based on specific conditions. To determine the best feature splits in decision trees, various measures of impurity are used. One commonly used measure, known as Gini impurity (17), quantifies the probability of misclassifying a randomly chosen element in a given node. It is defined by Equation 4.1.

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2 \tag{4.1}$$

In equation 4.1, D represents a dataset containing samples from k classes, and p_i represents the probability of a sample belonging to class i at a given node. The formula calculates the impurity or 'disorder' of a node by summing the squared probabilities for each class label. Lower values of Gini impurity indicate a more homogeneous distribution of class labels within a node, which leads to better splits for decision tree construction. The final predictions are made by the terminal nodes or leaf nodes. However, individual decision trees often have limited accuracy, especially when they become deep with numerous splits, which can result in overfitting (18).

To address the limitation of individual decision trees, Random forests introduce the concept of 'bagging' to construct an ensemble of decision trees. Bagging involves generating multiple bootstrap samples from the original training dataset. Each bootstrap sample is created by randomly drawing observations from the training data with replacement, resulting in a sample of the same size, with potentially some duplicate instances. By training decision trees on different bootstrap samples, random forests induce diversity within the ensemble and thereby reduce variance and the tendency to overfit.

In addition to bagging, random forests introduce randomness at the feature level. At each node of a decision tree, instead of considering all available features, only a random subset of features is considered for determining the best split. This randomness helps to decorrelate the decision trees within the random forest, making them more diverse and less prone to overfitting. By randomly selecting a subset of features at each split, random forests can effectively handle high-dimensional data and capture complex interactions between features. Furthermore, this feature randomness enables the estimation of feature importance, as the collective performance of the random forest reveals the relative importance of different features in the prediction process.

After constructing all the decision trees within the random forest, predictions are made by aggregating the outputs of the individual trees. In the case of classification problems, a common approach is to employ majority voting. This involves counting each tree's prediction, and the class with the highest count is selected as the final prediction. On the other hand, for regression problems, the individual tree predictions are typically averaged to obtain the final prediction. This ensemble voting technique allows the random forest to benefit from the collective knowledge of the trees, mitigating the impact of individual tree errors or biases and resulting in more accurate and reliable predictions.

To provide a visual representation of the Random Forest algorithm, Figure 4.1 illustrates the structure of a random forest.

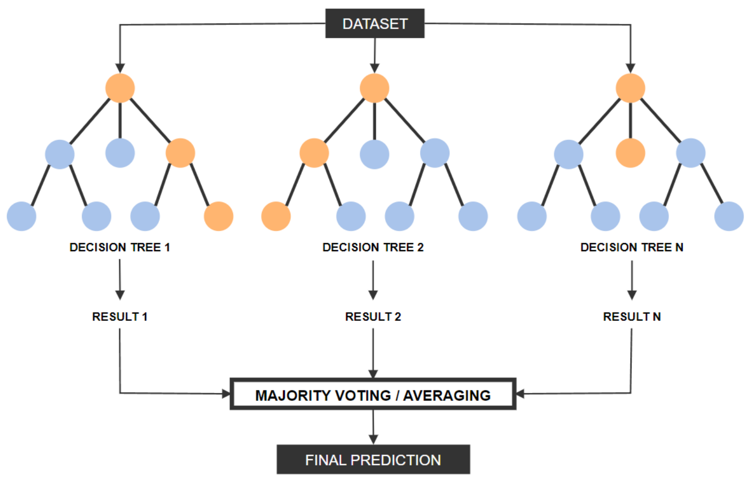


Figure 4.1: General structure of a Random Forest.

A valuable aspect of random forests is feature importance. Feature importance in random forests is a technique that assesses the significance of different features in making predictions. It measures the contribution of each feature in reducing impurity or improving predictive accuracy. Common methods include Mean Decrease Impurity (MDI) and Mean Decrease Accuracy (MDA) (19).

4.1.2 Extreme Gradient Boosting

Extreme Gradient Boosting, better known as XGBoost, was introduced in 2016 by Chen et al. (20). The XGBoost algorithm is an advanced implementation of the gradient boosting algorithm. This section provides a theoretical understanding of the Gradient boosting algorithm.

At its core, XGBoost utilizes decision trees (previously discussed in section 4.1.1) and the gradient boosting technique. In XGBoost, decision trees serve as the base models for the ensemble. However, individual decision trees are typically weak models with limited predictive power. To overcome the limitations of weak models, XGBoost employs gradient boosting. This technique involves training decision trees sequentially, where each subsequent tree aims to correct the errors made by the previous trees. Each tree learns from its predecessors and focuses on updating the residual errors, which are the differences between the predicted and actual values. By iteratively improving the model's predictions through the ensemble of decision trees, XGBoost achieves higher predictive accuracy.

In XGBoost, a specific objective function needs to be optimized during training. The objective function consists of two components: a loss function that measures the discrepancy between predicted and actual values, and a regularization term that discourages complex models in order to prevent overfitting. The loss function quantifies the error and guides the model towards minimizing it. Different loss functions can be used depending on the type of problem. The regularization term helps control the complexity of the model, preventing it from becoming overly complex and improving generalization to unseen data.

XGBoost employs gradient descent optimization to iteratively minimize the objective function. It calculates the gradient and the Hessian (the second derivative) of the loss function with respect to the predicted values. These derivatives provide information about the direction and rate of improvement in each iteration. The model parameters are updated based on these derivatives, leading to a gradual improvement in prediction accuracy. During each iteration, XGBoost constructs decision trees based on the gradients and Hessians. The trees are built to approximate the negative gradients, which represent the direction of steepest descent in the objective function. The tree construction process involves selecting optimal splits and determining the leaf node values that minimize the objective function.

Once all the trees are constructed, XGBoost aggregates their predictions to make the final prediction. The ensemble prediction is typically a combination of the predictions from all the trees, weighted by their individual contributions. Additionally, XGBoost provides a measure of feature importance, indicating the relative importance of different features in making predictions. It calculates the total gain or total cover of each feature across all the trees, reflecting their contribution to the model's performance. This feature importance analysis helps in understanding the relevance and impact of different features in the prediction process.

4.1.3 Neural Network

Neural networks (21), also known as artificial neural networks, are a class of machine learning models inspired by the structure and functioning of the human brain. They consist of interconnected layers of nodes called neurons, organized into an input layer, one or more hidden layers, and an output layer. Each neuron in a hidden layer receives inputs from the previous layer, applies a set of weights to those inputs, and passes the weighted sum through an activation function to produce an output. This process is repeated layer by layer until the output layer is reached, which produces the final predictions. The general structure of a neural network can be seen in Figure 4.2

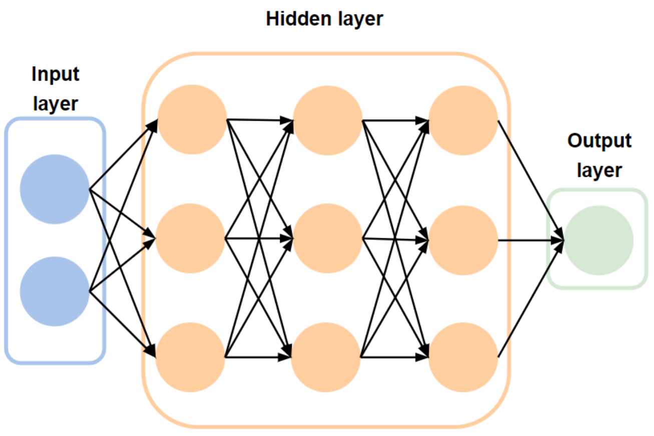


Figure 4.2: General structure of a neural network.

The training of a neural network involves optimizing the weights of the connections to minimize a defined objective function, often referred to as the loss or cost function. The objective function quantifies the discrepancy between the outputs of the network and the true outputs. The choice of loss function depends on the type of problem being solved, such as mean squared error for regression or cross-entropy loss for classification.

To optimize the weights, neural networks employ a technique called *backpropagation*((22), (23)). Backpropagation involves computing the gradients of the loss function with respect to the weights of the network's connections. These gradients indicate the direction and rate of improvement for each weight to minimize the loss. By iteratively updating the weights in the direction of the negative gradient, neural networks gradually improve their predictions.

The backpropagation process involves two main steps: forward propagation and gradient descent. During forward propagation, the input data is fed into the neural network, and the values are propagated through the layers. At each neuron, the weights associated with the incoming connections, along with an activation function, are applied to generate an output. The output of the last layer represents the predicted values or probabilities associated with the task at hand.

During gradient descent, the gradients of the loss function with respect to the weights are calculated using the chain rule. These gradients provide information about how the weights should be adjusted to minimize the loss. The weights are updated by taking small steps in the opposite direction of the gradients, with the step size controlled by a learning rate parameter. This iterative process continues until the network's performance reaches a satisfactory level or a specified number of training iterations is completed.

Once the neural network is trained, it can make predictions by feeding new input data through the network using the learned weights. The output of the network represents the predictions for the given input.

4.1.4 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of neural network designed to process sequential and temporal data. They are especially useful when the order of the input data matters, and the different elements in the sequence depend on each other. The key feature of RNNs is the ability to maintain internal memory storage, which allows them to capture information from previous inputs and use it to influence the processing of subsequent inputs. Memory storage is the reason why RNNs are great at predicting sequential patterns in data.

At the core of an RNN is the recurrent layer, which consists of recurrently connected nodes, also known as recurrent units. Figure 4.3 illustrates the distinction between a regular neural network, as shown in Figure 4.2, and a recurrent neural network. The most commonly used type of RNN cell is the Long Short-Term Memory (LSTM) (24) unit. The key innovation of LSTM networks is their ability to selectively retain or forget information from the past, allowing them to maintain long-term memory over extended sequences.

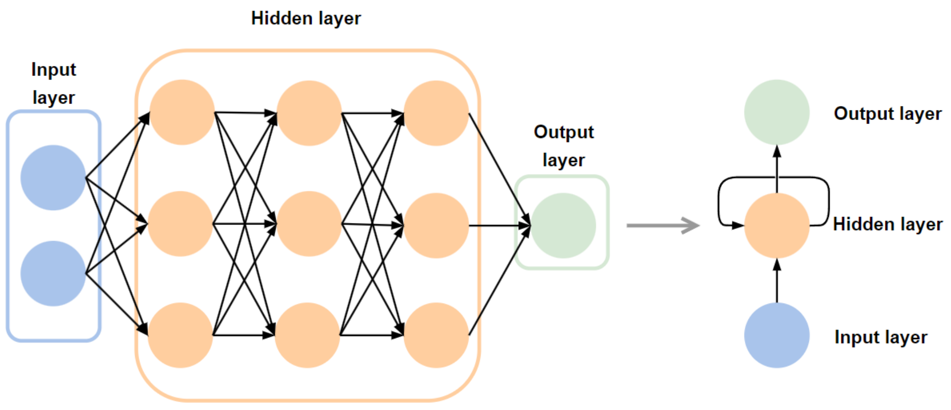


Figure 4.3: Structural difference between neural networks and recurrent neural networks.

The three main components of an LSTM cell are the input gate, the output gate, and the forget gate. These gates are designed to control the flow of information and determine what to remember, what to forget, and what to output at each time step. The entire structure of an LSTM unit can be seen in Figure 4.4, but it will also be explained in further detail.

- **Input Gate**: The input (25) determines which parts of the current input and previous hidden state are relevant and should be stored in the memory cell. It takes the current input and the previous hidden state as inputs and passes them through a sigmoid activation function that determines how much of the new information gets stored in the memory cell.

- **Forget Gate**: The forget gate (26) decides which information from the previous memory cell state should be discarded. Like with the input gate, the information given by the current input and the previous hidden state get passed through a sigmoid activation function that controls the amount of information that should be forgotten.
- **Memory Cell**: The memory cell updates and stores information over time by combining the input and previous hidden state, adjusted by activation functions.
- **Output Gate**: The output gate determines what information from the current memory cell state should be output as the hidden state at the current time step. It takes the current input and the previous hidden state as inputs and applies a sigmoid activation function. The output gate activation is then combined with the current memory cell state, modified by a tanh activation function, to produce the current hidden state output.

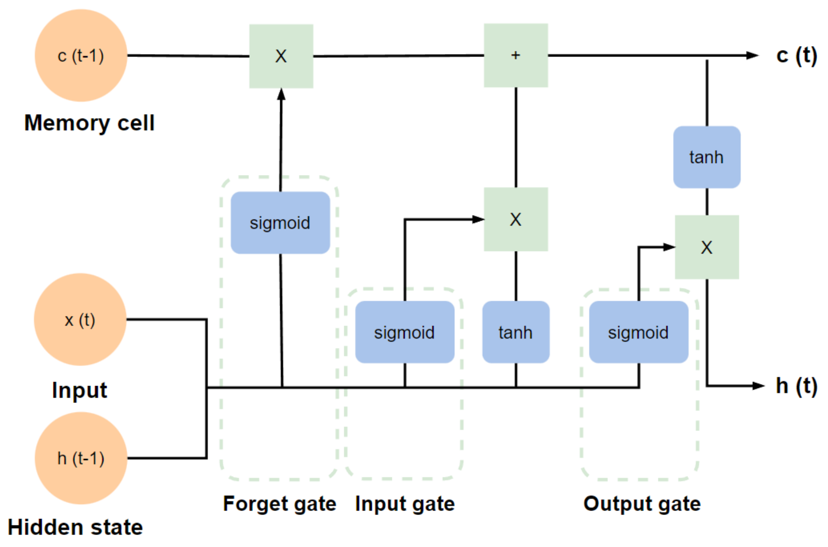


Figure 4.4: General structure of an LSTM unit.

During training, LSTM networks use variants of backpropagation, such as *backpropagation through time* (BPTT) (27), to optimize their parameters. The gradients are calculated through time and used to update the weights and biases of the LSTM cells, enabling the network to learn to model complex sequential patterns.

4.2 Data sampling techniques

As mentioned in section 3.4, the data exhibits a severe class imbalance. Class imbalance can challenge machine learning algorithms, as they tend to favour the majority class and struggle to capture patterns in the minority class. To mitigate the effects of class imbalance and improve the model's performance, multiple sampling techniques were employed in this study. Specifically, *SMOTE* (Synthetic Minority Over-sampling Technique), Borderline SMOTE, and undersampling techniques were applied to the data.

SMOTE SMOTE, introduced by Chawla et al. (28), is a widely used technique for handling imbalanced datasets. It generates synthetic instances for the minority class by interpolating between neighbouring instances. The synthetic instances are created by randomly selecting a minority class instance and then randomly selecting one or more of its k-nearest neighbours. A new instance is synthesized by taking a linear combination of the feature vectors of the selected instance and its neighbours.

By applying SMOTE, the dataset is augmented with synthetic instances, effectively balancing the class distribution. This technique allows the model to learn from a more balanced representation of the data, improving its ability to capture the patterns and characteristics of the minority class.

Borderline SMOTE Borderline SMOTE is an extension of the SMOTE technique that focuses on the borderline instances of the minority class. Borderline instances are those that are located near the decision boundary, making them more difficult to classify accurately. Borderline SMOTE specifically targets these instances to create synthetic samples (29).

By giving special attention to the borderline instances, Borderline SMOTE aims to enhance the discriminative ability of the model. It generates synthetic instances in a similar manner to SMOTE, but with a greater emphasis on the borderline instances. This technique helps in capturing the complex decision boundaries between classes and can potentially improve the model's performance, especially in scenarios with severe class imbalance.

Undersampling Undersampling is another technique employed to handle class imbalance. It involves randomly selecting a subset of instances from the majority class to balance the class distribution (30). By reducing the number of instances from the majority class, undersampling aims to create a more balanced representation of the data.

Undersampling can help prevent the model from being overwhelmed by the majority class, allowing it to focus more on the minority class and improve its ability to capture its characteristics. However, undersampling also comes with the risk of losing valuable information present in the majority class instances. Hence, a careful balance must be struck to avoid excessive loss of information while effectively addressing the class imbalance.

4.3 Evaluation techniques

Evaluating a model with a severe class imbalance requires specific care. Due to the nature of certain metrics, they may not be suitable for direct utilization in such scenarios. However, in this research, a set of evaluation metrics has been carefully selected to address the challenges posed by imbalanced datasets. These metrics provide a comprehensive assessment of the model's performance in shot detection tasks while accounting for class imbalance. The evaluation metrics used in this research are described below.

Precision Precision represents the proportion of correctly predicted positive instances (TP: shots) out of all instances predicted as positive. It indicates the accuracy of positive predictions. In the context of shot detection, precision measures how well the model identifies shots without generating too many false alarms (false positives, FP). A higher precision indicates a lower number of false positives and thus a more reliable shot detection system. The formula for precision is: Precision $= \frac{TP}{TP+FP}$

Recall Recall, also known as sensitivity or true positive rate, represents the proportion of correctly predicted positive instances (TP: shots) out of all actual positive instances. It focuses on capturing all positive instances without missing any (false negatives, FN). In shot detection, recall measures the ability of the model to detect all shots present in the data. A higher recall indicates a lower number of missed shots. The formula for recall is: Recall $= \frac{TP}{TP+FN}$

F1 score The F1 score combines precision and recall into a single metric, providing a balanced measure of the model's performance. It is the harmonic mean of precision and recall. A higher F1 score indicates a better trade-off between precision and recall. The formula for the F1 score is: F1 score $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

The F1 score is particularly useful when dealing with class imbalance because it considers both false positives and false negatives, ensuring a balanced evaluation of the model's performance.

AUC-ROC AUC-ROC (Area Under the Receiver Operating Characteristic Curve) summarizes a binary classification model's overall performance. It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. A higher AUC-ROC value indicates a better-performing model with a higher true positive rate and a lower false positive rate. It provides a comprehensive evaluation of the model's ability to distinguish between positive and negative instances.

Tolerance interval As described in section 3.2, a shot only gets one label in the data, while a shot in reality takes up way more than one frame. For this reason, a so-called 'tolerance interval' was used when evaluating the performance of the models. A tolerance interval allows for a range or tolerance around the predicted shot boundaries, considering the inherent variability in shot annotations. It provides a measure of flexibility by accepting predictions that are close to the true shot boundaries, even if they are not an exact match. In other words, shots will also be considered correctly detected if they are within a certain marge from the actual label. This approach acknowledges the practical challenges in precisely labelling shots and enables a more realistic evaluation of model performance.

Chapter 5

Results

This chapter will describe the results achieved by the methods previously described in chapter 4.

5.1 Experimental setup

Several experiments were conducted using the methods mentioned in Chapter 4. For these experiments, the data was divided into train, validation, and test sets using a 70/20/10 split. Each of the four methods was tested multiple times, employing various sampling techniques described in Section 4.2. The evaluation was performed using the metrics outlined in Section 4.3, resulting in four distinct results for each model.

Tree ensemble parameters The hyperparameters of the tree ensembles, Random Forest and XGBoost, were tuned using grid search. The tuned hyperparameters for the tree ensembles can be seen in Table 5.1.

Random Forest

	No sampling	SMOTE	Borderline SMOTE	Undersampling
Estimators	200	300	200	150
Max depth	None	None	None	None
Max features	sqrt	sqrt	sqrt	sqrt
Min sample split	5	5	5	5
Class weight	10	5	5	5

XGBoost

	No sampling	SMOTE	Borderline SMOTE	Undersampling
Estimators	100	300	300	100
Max depth	5	10	10	10
Learning rate	0.01	0.1	0.1	0.001
Class weight	1	1	1	1

Table 5.1: Hyperparameters for tree ensembles in experiments.

During the tuning process, a decision had to be made regarding the model's objective: whether to maximize the detection of shots or minimize the inclusion of noise (false positives). After consulting with Forward Football, it was determined that minimizing noise detection should get priority, rather than attempting to detect all shots. Therefore, the evaluation scoring used for the grid search results was precision, with a 3-fold *cross-validation* scheme (31). For the application of SMOTE, a sampling strategy of 0.5 was employed, resulting in the minority class being approximately half the size of the majority class after resampling. This strategy was chosen based on empirical research.

Neural network structure Through empirical research, the neural network structure was carefully determined by conducting numerous small experiments. These experiments aimed to optimize various components such as the number of nodes, layers, activation functions, class weights, learning rates, and batch sizes. It was crucial to strike a balance between complexity and generalization when devising the network architecture.

Ultimately, the chosen neural network comprises three dense layers with respective neuron counts of 256, 128, and 64. This design, along with a significant number of neurons, is well-suited for capturing intricate relationships within the data. However, it's important to exercise caution as larger layers carry an increased risk of overfitting, where the model becomes too specialized to the training data and performs poorly on new inputs.

To introduce non-linearity and facilitate the model's ability to learn complex patterns, the activation functions relu, tanh, and sigmoid were selected. Relu activations are a widely used choice for introducing non-linear transformations, aiding the model in capturing intricate patterns. Tanh activation further enhances the model's capability to learn non-linear relationships involving both positive and negative values. Finally, the sigmoid activation function maps the output to a range between 0 and 1, making it suitable for binary classification tasks.

Overall, the network architecture was intentionally kept relatively simple to mitigate the risk of overfitting. This decision helps strike a balance between complexity, which enables capturing complex relationships, and generalization, which ensures the model's ability to perform well on new and unseen data.

Recurrent neural network structure The RNN architecture is designed with two LSTM layers to effectively capture long-term dependencies in sequential data. The first LSTM layer has 64 neurons, followed by the second layer with 32 neurons. The output layer is a dense layer with a single neuron and a sigmoid activation function, which is used to transform the output for binary classification. This simple architecture aims to prevent overfitting while maintaining a good balance between generalization and complexity. The model takes sequences that are 25 frames long (equivalent to 5 seconds) as input.

Evaluation For the evaluation of the models, a tolerance interval of 2 seconds (10 frames) was chosen. This interval was chosen based on the domain knowledge of individuals within the host organization, who said that if the shot was detected about two seconds before or after the actual shot, the information would still be valuable for Forward Football. Besides, the party that delivered the data sometimes detects the shot with a few seconds delay, meaning the actual shot might be happening 1 or 2 seconds before or after the label.

For one match, the shots were manually labelled, also referred to as the 'ground truth,' in order to assess the models' ability to accurately detect all the true shots in the data. This evaluation helps determine how closely the models align with the actual shots, rather than relying on potentially erroneous shots initially detected by a third party.

5.2 Baseline results

To evaluate the quality of the model's results, it is essential to establish a baseline for comparison. In the absence of comparable prior research, a randomness baseline was employed. This baseline randomly determines whether a shot occurs or not in the validation and test sets. By adopting this approach, we can assess whether other models perform better, worse, or at the same level as random guessing.

Table 5.2 presents the outcomes of the random baseline. The recall of 0.495 indicates that approximately half of the shots were correctly identified at their reported timestamps. However, the precision of 0.004 suggests a large number of false positive cases in the results. In other words, a lot of shots that did not occur were detected, also known as noise.

Furthermore, the AUC-ROC score of 0.499 confirms that the model's performance is comparable to random guessing. As expected, a random baseline lacks any meaningful predictive capability.

Precision	**Recall**	**f1-score**	**AUC-ROC**
0.004	0.551	0.008	0.525

Table 5.2: Baseline results.

5.3 Random Forest results

The test set results of the Random Forest model can be found in Table 5.3. The results of the validation set can be found in Appendix A.1.

	Precision	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.3	0.188	0.231	0.593
SMOTE	0.207	1.0	0.343	0.795
Borderline SMOTE	0.237	0.646	0.346	0.746
Undersampling	0.043	1.0	0.083	0.933

Table 5.3: Random Forest results on the test set.

Table 5.3 shows the performance of different models. All models achieve a precision ranging from 0.2 to 0.3, indicating that approximately 70-80% of the shots they detect are actually noise, while only 20-30% are correctly identified shots. The undersampling model performs poorly in comparison, suggesting that it detects a significant number of false positives. This could be because important patterns were removed when undersampling the majority class.

In terms of recall values, the no-sampling model only captures 18.8% of the shots in the dataset. While it identifies less noise, it also fails to detect most of the shots in the test set. Considering the low F1-score and AUC-ROC score, it seems that the model learned very little. This is not surprising since it learned from an extremely imbalanced dataset without any sampling techniques, relying solely on class weights.

Both the SMOTE and undersampling models achieve a recall of 1.0, indicating that they detect all shots in the test set. However, when considering recall in conjunction with precision, it becomes evident that the undersampling model detects all shots not because it has learned the pattern, but because it detects a large number of shots, accidentally including the real shots as well. In contrast, the SMOTE and borderline SMOTE models strike a better balance between detecting shots and avoiding noise.

Overall, the SMOTE and borderline SMOTE models perform the best among the models, with the borderline SMOTE model slightly outperforming the regular SMOTE model. All models outperform the randomness baseline, although the improvement with the undersampling model is negligible. It is also worth noting that all models exhibit slightly worse performance on the test set compared to the validation set.

Analyzing the feature importance of the borderline SMOTE model reveals that the most influential feature is the future distance to the goal, followed by the future speed and whether the direction of the ball is toward the goal or not. Clearly, the movement of the ball after the shot has occurred proves to be the most critical factor in detecting shots, aligning with the real-world understanding of when a shot will happen.

To delve deeper into the performance of the borderline SMOTE model, its performance on the ground truth was examined. When applied to the ground truth, the model achieves a precision of 0.221, indicating that approximately 78% of the detected shots are false positives. However, the recall value is 1.0, meaning that all shots are detected, resulting in an overall F1-score of 0.362. Surprisingly, the performance of the model on the ground truth actually surpasses its performance on the test set by a small margin.

To further analyze the model's performance based on the ground truth, the detected shots were examined using video footage of the match. This allowed for a more detailed understanding of where the model struggled in accurately detecting shots. Upon reviewing the footage, it was observed that approximately half of the false positives were passes to the goalkeeper or other types of passes near the goal. These passes could potentially be eliminated if ball possession data were available. Additionally, around 20% of the false positives were shots that fell just outside the defined tolerance interval and were consequently not filtered out. Lastly, approximately 30% of the detected shots were determined to be nonsensical, such as the ball rolling out of bounds along the sidelines or backlines.

5.4 XGBoost results

The test set results, presented in Table 5.4, provide insights into the performance of the random forest model using different sampling techniques. The results of the validation set can be found in Appendix A.2.

	Precision	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.0	0.0	0.0	0.5
SMOTE	0.068	1.0	0.128	0.851
Borderline SMOTE	0.162	1.0	0.278	0.793
Undersampling	0.033	1.0	0.064	0.908

Table 5.4: XGBoost results on the test set.

In reference to the results presented in Table 5.4, an immediate observation is that the model without any sampling techniques exhibits a precision, recall, and F1-score of 0.0, indicating an inability to detect any shots. This model failed to learn any discernible shot patterns, resulting in a performance that is arguably worse than random guessing. On the other hand, all sampling techniques achieved a recall of 1, indicating the successful detection of all shots. However, when examining the precision values, it becomes evident that these models not only detected true positives (shots) but also produced numerous false positives or noise, which affected their recall performance. The models went to extreme measures to detect the minority class, suggesting a misalignment between the sampling techniques and the model's implementation.

Looking specifically at the undersampling model, the precision of 0.033 suggests a high detection of noise. Although it also managed to identify all shots, the F1-score of 0.064 indicates only slight improvement over the randomness baseline. It is likely that the undersampling process caused the loss of underlying shot patterns.

Overall, similar to the random forest model, the borderline SMOTE models exhibited the best performance. All models outperformed the randomness baseline, except for the model without any sampling techniques. However, all models displayed significantly poorer performance on the test set compared to the validation set.

Additionally, examining the feature importance of the borderline SMOTE model revealed that, according to XGBoost, the most influential features by a significant margin were the future distance to the goal and the future direction towards the goal. Most other features were comparatively not significant.

Furthermore, an analysis of the borderline SMOTE model was conducted by evaluating its performance on the ground truth. It achieved a precision of 0.114, which is lower than the precision obtained on the test set (0.162). Additionally, the recall value of 1.0 indicates successful detection of all shots, but at the expense of a substantial amount of noise, as indicated by the precision. Overall, the F1-score of 0.204 shows that the model's performance on the ground truth was weaker than on the test set.

Upon reviewing the video footage of the match and examining the false positive detections, it became evident that a significant majority, more than half, of the false positive shots were actually passes made to the goalkeeper or other passes near the goal. A smaller portion of the false positives included instances where the ball was rapidly approaching the goal but a shot had not yet been taken, falling outside the defined tolerance interval. Additionally, a remaining portion of the false positives appeared to be attributed to the ball rolling out of bounds.

5.5 Neural Network results

The test set results of the neural network can be found in Table 5.5. The results of the validation set can be found in Appendix A.3.

	Loss	**Precision**	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.038	0.215	0.896	0.347	0.744
SMOTE	0.058	0.166	1.0	0.284	0.845
Borderline SMOTE	0.065	0.177	1.0	0.301	0.814
Undersampling	0.04	0.268	0.938	0.416	0.787

Table 5.5: Neural Network results on the test set.

Table 5.5 provides insights into the models' performance. Firstly, it is evident that all models achieve a relatively low cross-entropy loss, indicating successful minimization of the discrepancy between predicted probabilities and actual binary labels. Examining precision, all models demonstrate precision values ranging from 0.15 to 0.3, implying that 15% to 30% of the detected shots are actual shots, while the remaining detections are noise. Interestingly, all models detect almost all of the shots in the test set, with the SMOTE and borderline SMOTE models achieving a recall of 1.0. However, it is worth noting that these models also exhibit the lowest precision among the models, suggesting that they might have also detected more noise compared to other models.

An intriguing observation is the performance of the network without any sampling technique. It performs comparably, if not better, than some models that addressed class imbalance. This suggests that the neural network effectively captures the underlying pattern that defines a shot, even without the assistance of sampling techniques.

Overall, the neural networks demonstrate good proficiency in identifying the underlying patterns that define a shot, with the undersampling model performing the best. Despite the potential information loss associated with undersampling, mitigating the bias towards the majority class appears to have benefited the model in this case. Importantly, all models outperform the randomness baseline, although their collective performance is slightly lower on the test set compared to the validation set.

To further explore the performance of the undersampling model, its effectiveness on the ground truth was assessed. When applied to the ground truth, the model achieves a recall of 1, successfully detecting all true shots. Additionally, it attains a precision of 0.333, implying that although some noise is still detected, the model demonstrates greater resilience to it compared to the previously discussed models. Overall, the model achieves an F1-score of 0.5, indicating that the undersampling model performs well on the ground truth.

Upon conducting further investigation into the false positive instances, it was discovered that the majority of them primarily involved passes occurring near the goal, either directed towards the goalkeeper by their own teammates or between other players. Additionally, a few cases were identified where shots were detected that fell just outside the defined tolerance interval, indicating that the ball was rapidly advancing towards the goal. Lastly, a small number of false positive instances were found to be attributed to nonsensical occurrences.

5.6 Recurrent Neural Network results

The test set results of the neural network can be found in Table 5.6. The results of the validation set can be found in Appendix A.4.

	Loss	**Precision**	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.055	0.128	0.354	0.188	0.558
SMOTE	0.064	0.043	0.083	0.057	0.507
Borderline SMOTE	0.056	0.044	0.083	0.058	0.497
Undersampling	2.436	0.004	1.0	0.007	0.539

Table 5.6: Recurrent Neural Network results on the test set.

Table 5.6 provides an analysis of the models' performance. The cross-entropy loss indicates that most models achieved similar results, except for the undersampling model, which performed significantly worse with a loss of 2.436. This high loss value indicates that the undersampling model struggled to distinguish shots from other observations in the test set. Moreover, the precision of the undersampling model further confirms its poor performance, as it detected less than 1% of true positives, suggesting an excessive number of false positives. This outcome might be attributed to the removal of recurrent patterns through random undersampling, leaving insufficient information for the model to learn from.

However, the other models did not perform significantly better. Both the SMOTE and borderline SMOTE models exhibited similar performance on the test set, with a recall of 0.083 and precisions of 0.043 and 0.044, respectively. The low recall values indicate that these models failed to identify the majority of actual shots, suggesting an inability to capture shot patterns effectively. Furthermore, the precision values indicate that although the models struggled to detect shots, they identified a considerable amount of noise. In contrast, the model without any sampling techniques outperformed the models with sampling techniques, most likely because the recurrent nature of the data remained undisturbed. However, even this model's performance was unsatisfactory, with a precision of 0.123 and a recall of 0.188. It detected less than half of the actual shots, and of the shots it did detect, over 80% were false positives.

Overall, the model without sampling techniques demonstrated the best performance, likely due to the preserved nature of the recurrent data. However, all models exhibited poor performance, with the no sampling and undersampling models performing slightly better than the randomness baseline. Surprisingly, the SMOTE and borderline SMOTE models even performed worse than the randomness baseline. It's worth noting that all models showed slightly lower performance on the test set compared to the validation set.

Upon evaluating the no sampling model's performance against the ground truth, its inadequacies are once again apparent. Despite its precision of 0.167, which indicates some ability to identify shots, the model still detected a significant amount of noise, although less than the previously discussed models. The recall value of 0.28 reveals that the model only captured less than a third of the true shots present in the dataset. Overall, the model's performance is reflected in its F1 score of 0.209, which, although surpassing the random baseline, still demonstrates its suboptimal performance.

Investigating the false positive detections for the best models posed a challenge due to the high number of instances flagged by the model. Upon conducting a general investigation, it appeared that the model tended to classify any movement of the ball towards the goal as a shot.

Chapter 6

Discussion

Limitations This research, like any other, has certain limitations that should be acknowledged. One of the major limitations is the absence of player ball possession information in the data. This absence significantly impacts the models' performance and contributes to the noise detected. Including features indicating whether the player in possession of the ball is on their own half or the opponent's half could greatly reduce false detections. Intuitively, a shot only occurs when the player with the ball is on the opponent's side. As discussed in Section 5, many incorrectly detected shots are actually passes to the goalkeeper. If the model had knowledge of the player's ball possession and their location on the field, these instances would not be falsely identified as shots. While attempts were made to extract ball possession information (as described in Section 3.2), it is impossible to confirm the accuracy of the algorithm's labelling. The third-party provider of the data has access to ball possession information but chooses not to disclose it to Forward Football, as sharing such information could potentially lead to the replication of their algorithms.

Another limitation of this research is the precision of the coordinates. As mentioned in Section 3.1, the data provides meter-accurate coordinates, resulting in a grid of 1 meter by 1 meter on the pitch. Consequently, even a slight movement of the ball from one grid to another is recorded as a full-meter displacement, whereas in reality, it might have only moved 30 cm. The lack of precise coordinates hampers the accuracy and reliability of the extracted features. If more precise coordinate data were available, it is possible that the extracted features would also be more accurate.

Similarly, the current data frequency of 5 Hz may not be optimal. The influence of data frequency on the results remains uncertain, as the frequencies used in existing literature vary, but they are typically higher than 5 Hz. It is unclear to what extent the lower frequency used in this research may impact the findings. Further exploration and comparison with different data frequencies could provide valuable insights into the optimal frequency for detecting shots accurately.

Future research In the realm of future research, there are numerous intriguing possibilities to explore. One avenue of interest involves delving deeper into the effects of different neural network structures, such as variations in the number of layers, nodes, and activation functions. Additionally, investigating the impact of varying the number of timesteps used in recurrent neural networks could yield valuable insights. Furthermore, it would be worthwhile to explore the potential of entirely new algorithms like logistic regression or support vector machines. Alongside these algorithmic investigations, it would be beneficial to explore alternative sampling techniques, such as random oversampling, ADASYN, SMOTE-ENN, or SMOTE-Tomek, among others.

Exploring a rule-based detection approach instead of relying solely on machine learning models can be valuable. By employing a rule-based method, extensive control over the detection process is achieved, allowing the definition of specific criteria for distinguishing shots from other actions. This departure from machine learning approaches presents distinct advantages and opportunities for customization.

If none of the aforementioned ideas yield significant improvements in detection, an alternative approach could involve conducting an "after analysis" using logic rules to filter out false positive instances. By applying logical rules post-detection, it becomes possible to refine the results and reduce the number of erroneous classifications. This approach serves as a complementary step to the detection process, allowing for more precise and accurate identification of true positives while mitigating the impact of false positives. Implementing logical rules in the post-analysis stage can enhance the overall effectiveness and reliability of the detection system.

Chapter 7

Conclusion

Throughout this study, our research aimed to address the following question:

To what extent can machine learning be employed for the detection of football shots?

The dataset was evaluated using four different models, each employing different sampling techniques. These models displayed varied responses to the data. Notably, the tree ensembles (random forest and XGBoost) achieved the highest performance when utilizing the borderline SMOTE sampling technique, benefiting from their inherent ability to handle imbalanced data and noise robustness. However, without any sampling technique, the tree ensembles struggled to identify anything beyond non-shots.

Conversely, the neural network and recurrent neural network performed equally well or even better when no sampling techniques were applied. This suggests that neural networks possess superior adaptability to the data, potentially explaining their relatively stronger performance. The recurrent neural network, in particular, achieved the best results when no sampling technique was used, indicating the importance of preserving the data's inherent recurrent nature.

On the other hand, overall undersampling emerged as the least effective technique across all algorithms, except for the neural network, where it surprisingly outperformed other approaches. The negative impact of undersampling on the tree ensembles can be attributed to the loss of diverse data samples, which hindered their predictive accuracy. Undersampling primarily aimed to address bias towards the majority class but did not significantly contribute to detecting patterns in the minority class. Additionally, the undersampling technique likely disrupted the recurrent nature of the data, leading to subpar performance in the recurrent neural network.

Overall, all models faced challenges in noise detection, with each model producing at least twice as many false positives as true positives. However, with the exception of two models, all models demonstrated improvement over random guessing, albeit some to a minor extent.

Upon analyzing the false positives generated by the models, several distinct patterns in their mistakes became evident. Firstly, the models frequently misidentified passes that occurred in close proximity to or towards the goal, including passes directed towards the goalkeeper. Additionally, the models often detected the ball's motion towards the goal, occasionally identifying goals just outside the 2-second tolerance window. Finally, the models occasionally erroneously identified other miscellaneous ball actions, such as the ball accidentally rolling out of bounds.

In analyzing the feature importance, as indicated by the tree ensembles, it becomes evident that the future trajectory of the ball plays a pivotal role in detecting shots. Specifically, the models assign high importance to the future distance between the ball and the goal, as well as the future direction of the ball towards the goal. On the other hand, features such as the stationary direction of the ball (when the ball is essentially inactive for a brief period) or the past trajectory of the ball are deemed less significant and have lower importance in the models' decision-making process.

To address the research question, *To what extent can machine learning be employed for the detection of football shots?* It can be concluded that machine learning can be effectively utilized for shot detection. However, it is crucial to consider the data structure, model selection, and sampling techniques as they have a significant impact on the results.

Among the models examined in this research, the neural network with undersampling emerged as the most successful, achieving the highest F1 score of 0.416. This outcome suggests that employing undersampling as a technique in conjunction with a neural network can yield favourable results for football shot detection.

From a company perspective, the results are satisfactory, surpassing initial expectations for what could be achieved through a machine learning approach. However, the current model still generates numerous false positive observations, making it unsuitable for direct use by Forward Football. Efforts are underway to enhance the results by leveraging future research, as outlined in Chapter 6. Unfortunately, due to limited information, such as ball possession data, significant improvements to the results are currently unattainable.

Bibliography

[1] Charles Reep and Bernard Benjamin. **Skill and chance in association football**. *Journal of the Royal Statistical Society. Series A (General)*, **131**(4):581–585, 1968. 3

[2] Ian M Franks and Gary Miller. **Eyewitness testimony in sport**. *Journal of sport behavior*, **9**(1):38, 1986. 3

[3] Michael Horton, Joachim Gudmundsson, Sanjay Chawla, and Joël Estephan. **Automated classification of passing in football**. In *Advances in Knowledge Discovery and Data Mining: 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part II*, pages 319–330. Springer, 2015. 3

[4] Abdullah Khan, Beatrice Lazzerini, Gaetano Calabrese, Luciano Serafini, et al. **Soccer event detection**. In *International Conference on Image Processing and Pattern Recognition (IPPR)*, **3**, 2018. 3

[5] Haohao Jiang, Yao Lu, and Jing Xue. **Automatic soccer video event detection based on a deep neural network combined cnn and rnn**. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 490–494. IEEE, 2016. 3

[6] Xinyu Wei, Patrick Lucey, Stephen Vidas, Stuart Morgan, and Sridha Sridharan. **Forecasting events using an augmented hidden conditional random field**. In *Computer Vision–ACCV 2014: 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1-5, 2014, Revised Selected Papers, Part IV 12*, pages 569–582. Springer, 2015. 4

[7] Keven Richly, Max Bothe, Tobias Rohloff, and Christian Schwarz. **Recognizing Compound Events in Spatio-Temporal Football Data.** In *IoTBD*, pages 27–35, 2016. 4, 6

[8] Keven Richly, Florian Moritz, and Christian Schwarz. **Utilizing artificial neural networks to detect compound events in spatio-temporal soccer data**. In *Proceedings of the 2017 SIGKDD Workshop MiLeTS, Halifax, NS, Canada*, pages 13–17, 2017. 4, 6

[9] Emily E Cust, Alice J Sweeting, Kevin Ball, and Sam Robertson. **Classification of Australian football kick types in-situation via ankle-mounted**

inertial measurement units. *Journal of Sports Sciences*, **39**(12):1330–1338, 2021. 4

[10] Vasanth Tovinkere and Richard J Qian. **Detecting semantic events in soccer games: Towards a complete solution**. In *IEEE International Conference on Multimedia and Expo, 2001. ICME 2001.*, pages 212–212. IEEE Computer Society, 2001. 4

[11] Ferran Vidal-Codina, Nicolas Evans, Bahaeddine El Fakir, and Johsan Billingham. **Automatic event detection in football using tracking data**. *Sports Engineering*, **25**(1):18, 2022. 4

[12] Thierry Blu, Philippe Thévenaz, and Michael Unser. **Linear interpolation revitalized**. *IEEE Transactions on Image Processing*, **13**(5):710–719, 2004. 8

[13] David Harris and Sarah L Harris. *Digital design and computer architecture.* Morgan Kaufmann, 2010. 8

[14] Murali Shanker, Michael Y Hu, and Ming S Hung. **Effect of data standardization on neural network training**. *Omega*, **24**(4):385–397, 1996. 8

[15] Tin Kam Ho. **Random decision forests**. In *Proceedings of 3rd international conference on document analysis and recognition*, **1**, pages 278–282. IEEE, 1995. 13

[16] Leo Breiman. **Random forests**. *Machine learning*, **45**:5–32, 2001. 13

[17] Leo Breiman. *Classification and regression trees.* Routledge, 2017. 13

[18] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, **2**. Springer, 2009. 14

[19] Hong Han, Xiaoling Guo, and Hua Yu. **Variable selection using mean decrease accuracy and mean decrease gini based on random forest**. In *2016 7th ieee international conference on software engineering and service science (icsess)*, pages 219–224. IEEE, 2016. 15

[20] Tianqi Chen. **Story and lessons behind the evolution of XGBoost**, 2016. 15

[21] Pedro Domingos. **A few useful things to know about machine learning**. *Communications of the ACM*, **55**(10):78–87, 2012. 16

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. **Deep learning**. *nature*, **521**(7553):436–444, 2015. 16

[23] Michael A Nielsen. *Neural networks and deep learning*, **25**. Determination press San Francisco, CA, USA, 2015. 16

[24] Sepp Hochreiter and Jürgen Schmidhuber. **Long short-term memory**. *Neural computation*, **9**(8):1735–1780, 1997. 17

[25] Sepp Hochreiter and Jürgen Schmidhuber. **LSTM can solve hard long time lag problems**. *Advances in neural information processing systems*, **9**, 1996. 17

[26] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. **Learning to forget: Continual prediction with LSTM**. *Neural computation*, **12**(10):2451–2471, 2000. 18

[27] Michael C Mozer. **A focused backpropagation algorithm for temporal**. *Backpropagation: Theory, architectures, and applications*, **137**, 1995. 18

[28] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. **SMOTE: synthetic minority over-sampling technique**. *Journal of artificial intelligence research*, **16**:321–357, 2002. 19

[29] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. **Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning**. In *Advances in Intelligent Computing: International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I 1*, pages 878–887. Springer, 2005. 19

[30] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. **Exploratory undersampling for class-imbalance learning**. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **39**(2):539–550, 2008. 19

[31] Daniel Berrar. **Cross-Validation.**, 2019. 22

Appendix A

Validation results

A.1 Random Forest

Table A.1 presents the validation results of the XGBoost model with different sampling techniques.

	Precision	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.38	0.181	0.245	0.566
SMOTE	0.2	1.0	0.333	0.768
Borderline SMOTE	0.321	0.914	0.475	0.719
Undersampling	0.04	1.0	0.076	0.907

Table A.1: Random Forest results on the validation set.

A.2 XGBoost

Table A.2 presents the validation results of the XGBoost model with different sampling techniques.

	Precision	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.5	0.029	0.054	0.509
SMOTE	0.061	1.0	0.115	0.852
Borderline SMOTE	0.153	1.0	0.265	0.818
Undersampling	0.031	1.0	0.061	0.9

Table A.2: XGBoost results on the validation set.

A.3 Neural network

Table A.3 presents the validation results of the neural network with different sampling techniques.

	Loss	**Precision**	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.045	0.222	1.0	0.365	0.755
SMOTE	0.068	0.157	1.0	0.272	0.823
Borderline SMOTE	0.077	0.178	1.0	0.302	0.815
Undersampling	0.05	0.257	1.0	0.409	0.77

Table A.3: Neural Network results on the validation set.

A.4 Recurrent neural network

Table A.4 presents the validation results of the recurrent neural network model with different sampling techniques.

	Loss	**Precision**	**Recall**	**F1-score**	**AUC-ROC**
No sampling	0.053	0.153	0.323	0.208	0.539
SMOTE	0.065	0.025	0.047	0.033	0.506
Borderline SMOTE	0.059	0.072	0.131	0.093	0.525
Undersampling	2.195	0.004	1.0	0.008	0.571

Table A.4: Recurrent Neural Network results on the validation set.