Universiteit Leiden

Master Computer Science

Predicting Successful Sequences in Soccer using Boosted LSTM Networks

Name:	Jimmy Drogtrop
Student ID:	S1526383
Date:	03/08/2021
Specialisation:	Computer Science and Advanced Data Analytics
1st supervisor:	Arno Knobbe
2nd supervisor:	Laurentius A. Meerhoff
3rd supervisor:	Stylianos Paraschiakos

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Soccer match data is becoming widely available, and as a result, new methods to quantify team positioning are found. These methods could potentially be used as features in prediction models to rate players or to find better tactics. However, most methods have seen limited validation of their effectiveness in prediction models. In this thesis, we use a large library of existing spatial features to identify dangerous ball possessions. Some of these features are sparse, and were previously unusable in our models. We show that using NaN-filling, we can successfully introduce sparse features to increase performance of different classification models, including (boosted) LSTMs. We also show that our models are at least capable of comprehending simple indicators for danger. Lastly, we present the best performance of our models, which shows that boosting can improve performance. The LSTMs did not outperform our baseline methods in the current setups, but we show that they could have the potential to become better in the future.

Contents

1 Introduction **5**

2 Related Work **6**
2.1 Soccer Data and Analysis . . . 6
2.1.1 Event Data . . . 6
2.1.2 Tracking Data . . . 7
2.2 Machine Learning . . . 8
2.3 Identifying Dangerous Sequences using LSTMs . . . 9

3 Data **9**
3.1 Event Data . . . 10
3.2 Tracking Data . . . 10
3.3 Feature Sets . . . 10
3.4 Sequences . . . 11

4 Methods **12**
4.1 Data Preparations . . . 13
4.1.1 Identifying and Labelling Sequences . . . 13
4.1.2 Handling NaNs . . . 15
4.1.3 Sampling . . . 15
4.1.4 Normalisation . . . 16
4.1.5 Sequence Length . . . 16
4.2 Baseline Methods . . . 17
4.2.1 Random Forest Classifier . . . 17
4.2.2 Naive Bayes . . . 17
4.2.3 Logistic Regression . . . 18
4.2.4 AdaBoost Classifier . . . 18
4.3 LSTM with AdaBoost . . . 18
4.3.1 LSTM . . . 18
4.3.2 Boosted LSTM . . . 20
4.4 Metrics . . . 20
4.4.1 Accuracy . . . 20
4.4.2 Precision and Recall . . . 20
4.4.3 Area under the ROC Curve . . . 21
4.4.4 F1 Score . . . 21
4.4.5 Average Precision . . . 22
4.4.6 Matthews Correlation Coefficient . . . 22

5 Experiments **22**
5.1 Experiment I - Additional Features . . . 22
5.1.1 Results - LSTMs . . . 23
5.1.2 Results - Baseline Methods . . . 23
5.1.3 Results - LSTMs vs Baseline Methods . . . 23
5.1.4 Discussion . . . 25
5.2 Experiment II - Dataset Full Representativeness . . . 26
5.2.1 Results . . . 27

5.2.2 Discussion 28
5.3 Experiment III - NaN-Filling 29
5.3.1 Results - LSTMs 29
5.3.2 Results - Baseline Methods 30
5.3.3 Results - LSTMs vs Baseline Methods 30
5.3.4 Discussion 30
5.4 Experiment IV - Sampling 31
5.4.1 Results 31
5.4.2 Discussion 32

6 Conclusion **33**

1 Introduction

In recent years, interest in the applicability of sport science and data science in the domain of soccer has increased. This results in larger quantities and higher quality datasets collected by clubs, leagues or specialised private companies. Consequently, using more and better data, we can develop new methods that allow us to better understand and analyse the game of soccer.

Event data consists of actions, generally annotated manually and in detail. Interesting results were achieved by using event data. Rudd [1], for instance, used a Markov Chain based expected goals method, which could present the probability of scoring from a given location on the pitch. Such methods were some of the first to create models on goal scoring and pass or shot probability, which can be helpful in winning matches. However, event data alone is often considered to lack context [2] in understanding more complex, dynamic game behaviour, especially on players not directly involved in an action. This context can be provided by tracking data, which contains the location for each player. Tracking data allows us to visualise spatial relationships between the positions of players and the ball. A well-known example of applied tracking data is used in the concept of spatial dominance [3, 4, 5]. This models the team in control (or the disputed surface) for specific areas on the pitch. In this thesis, we mainly use spatial features, derived from tracking data, as model input.

A problem with many complex, spatial methods is that their actual significance is often insufficiently scientifically validated [6] due to the limited number of similarly formatted competitive match datasets [3, 4, 7]. We use a collection of 167 features on a 302 match soccer dataset of the highest professional soccer competition in the Netherlands. If features are important for models in solving team performance related problems, this could indicate that these features are indeed indicators for key match behaviour. This could contribute feature validation to the literature, but it would be a by-product of our main goal.

In this thesis, we try to predict the occurrence of danger in series of actions. We select a time period, called the time interval, in which we potentially encounter actions that lead to danger. The time interval used are ballwin events, explained in Section 3.1, which can roughly be described as more certain ball possessions. To identify successful sequences, which are time intervals with corresponding data, we first need to define dangerous in the context of football. Many success definitions, goals for instance, would introduce an asymmetry in the data, as the number of unsuccessful sequences of actions would greatly surpass the number of successful sequences. This introduces a balancing problem for machine learning approaches. We have to make a tradeoff between the quality and informativeness of an event, and how often it occurs during a match [8]. Because shots occur far more often than goals, and based on the intuition that actions similar to shots are generally strong indicators for goal scoring opportunities, in this thesis, a sequence is identified as successful if it is dangerous according to the *danger* definition. Danger is better described in Section 3.1, but it roughly includes goals, misses, attempted saves and headers on goal. With danger as success definition, unsuccessful sequences still occur more far more often than successful sequences. We use a boosting algorithm and investigate undersampling and oversampling as methods to decrease the imbalance effect.

Typical analysis is limited to the pitch situation on given moments during the match, but as play emerges over time, it makes sense to use the short-term history in prediction models.

Because LSTMs were successful in other sequential tasks in the past [9], by using LSTMs, we might be able to better predict dangerous attacks. Long Short-Term Memory (LSTM) models, a type of recurrent neural network models, can use sequential data and learn to remember important information from previous time intervals to make better informed decisions, for instance in predicting dangerous sequences. They have been applied to different types of sport-related problems, for instance in capturing match-data [10] and predicting future player positioning [11], but have seen limited application in identifying successful attacks. As far as we know, LSTMs have not yet been used successfully and reliably to predict dangerous sequences with a success definition similar to Danger. We aim to use it to identify key team behaviour that decides team effectiveness. Potentially, such a model could be used to recognise which players contribute often to creating or closing danger, and in which situations. So, a successful model could be used to predict match outcome, rate players based on their importance in attacking or defending, to counter identified strengths of upcoming opponents, or choosing which players to put in your match selection.

In Section 2, we discuss how this thesis fits in the literature of machine learning and sequence classification, and we show methods and soccer features that are currently available. We also show how LSTMs are currently used in the area of soccer, and how machine learning is applied to the problem of identifying dangerous sequences. In Section 3, we describe the Eredivisie dataset, and the features we have available to that dataset. Section 4 describes our classification methods, the metrics that validate these classification methods. We also present the data processing methods, including NaN-filling and sampling methods. In Section 5, we present the experiment setup, results and discussion for each conducted experiment. The experiments show the performance impact of adding new features to our models, and the effect of introducing sampling methods. Finally, we draw our final conclusions on our work and highlight potential future work in Section 6.

2 Related Work

In this thesis, we use LSTMs to classify sequences, according to the danger definition. This topic is covered mainly by two science specialisations: sport science and computer science. Sport science generally involves either experimental, small-sided setups or finding data driven patterns to approach sports-related problems. Computer science includes the field of machine learning, to which LSTMs belong. We will discuss related work from both sciences.

2.1 Soccer Data and Analysis

In recent years, we see new, data-driven ways to implement tactical concepts in soccer and other sports [2]. Generally, the retrieved data is either Tracking data or Event data.

2.1.1 Event Data

Event data is a collection of actions, often annotated manually and described in detail. It is generally easier and cheaper to obtain, and consequently, available more widely. Previously, datasets used for performance analysis mainly relied on frequency distributions of certain game events [6]. Some pioneering work with this type of data were the Markov Chain based expected goals method by Rudd [1] and the expected goals (xG) method by Green [12]. They used pass

and goal frequency to derive goal probability as whether having the ball in a certain position will, possibly after a series of passes, terminate in a goal or loss of possession. These approaches simplify context on intermediate actions in a sequence, where the execution of one action greatly influences which following actions are attempted or even possible [13]. Therefore, Van Haaren and Honnosset [8] and Decroos et al. [13] grouped similar sequences of events, based on the type of actions performed and the order of those actions, to capture more (intermediate) action context. To stop the clusters of similar sequences from becoming too small, as very similar event sequences rarely occur [8], they decided to simplify the source and destination of an event by using global ball directions between actions. By simplifying context of the positioning of players or the source and target of sequences of actions, we can ultimately miss the nuances that decide the outcome of a sequence. Because oversimplification is often required for experiments with similar events, Goes et al. [3] state that tactical performance should not be seen as a chain of events alone, but as a management of space, time and individual actions, in order to approach more difficult problems. Most importantly, player positional context (space) is required to comprehend more complicated tactical concepts, like off-the-ball runs and space creation [2, 3].

2.1.2 Tracking Data

To gather player positional context, tracking data can be utilised. There are three common methods to capture tracking data: GPS, Local Precision Measurements (LPM), or optical tracking. GPS gives a rough location using satellites, LPM [14] accurately tracks players using a carried sensor and beacons around the field, and optical tracking uses cameras and computer vision to capture player position and context [8], for instance to retrieve body posture [15]. Compared to event data, tracking methods allow for more reliable capture of team positioning during matches [6].

Tracking data has previously been used to monitor physical performances or training loads [3, 16, 17], but has already been used to, automatically and more objectively, quantify team tactical weaknesses and player behaviour [2]. Concepts like I-Mov and D-Def [3] for example, are based on the intuition that teams have to create space and disrupt the defensive organisation to create scoring opportunities. The intuition that we should disrupt enemy lines is also mentioned by Fernández et al. [18] as breaking lines. They propose expected possession value (EPV) to evaluate the potential value of ball drives, shots or passes to any location. It returns a value between -1 and 1, expressing the likelihood of a possession ending in a goal for the attacking team opposed to a goal for the defending team. This way, EPV incorporates the risk of receiving a goal, but in this work, we are solely interested in the chance of scoring goals. Still, because scoring involves a lot of nuances [2], it is valuable to know that we can also measure how effectively we disorganise defensive positioning.

Modelling the factors involved in successfully creating space has been a major focus in the field. Spatial dominance [2, 3, 4, 5] captures context on pitch occupation and both on and off the ball movement. It can be used to find areas where the ball should have been passed, or a player should have been positioned. Peralta et al. [2] attempt to model the creation of space based on three factors: pass probability, impact, and control, or respectively, maximising probability of pass success to or from the player in possession, occupying dangerous areas on the pitch, and maximising the total amount of controlled space. These methods show that

how we perceive danger can be modelled, but it does not apply this knowledge to problems similar to predicting danger, which currently limits their potential. Link et al. [19] present a similar approach to modelling danger, where they consider danger as a single value aggregate of four factors; position on the pitch, control over the ball, received pressure and density in dangerous areas between the ball and the goal, which they call the interception zone. This method is probably closest related to our success definition of danger, but involves some human configuration between the four factors, where we prefer a data-driven approach.

As shown, some potentially important tactical concepts, like identifying important space or breaking lines, can already be captured. However, not all of them are directly usable in predicting dangerous sequences with single value data-driven features or as success definition, which may become a problem, as explained in Section 2.3. For such problems, simpler spatial features may be more valuable, and are arguably easier to implement in larger quantities. Team width and length [14, 20], surface area [14], as well as variations on the idea of centroids [14, 20], where we measure the average position of a set of players, have already had some success in predicting the winning team in a match. Some of these features are also used in our work. Section 3.2 presents a list of features currently available for machine learning in the Eredivisie dataset.

2.2 Machine Learning

The problem of predicting whether a sequence is dangerous is considered a sequential classification problem. Sequential classification is the task of categorising time-sensitive data, where data consists of feature values that exist over time. A popular approach to solving such problems involves deep-learning network. It requires expert domain knowledge to properly prepare these networks, and results are considered difficult to interpret, but it has achieved high performance in varying and complex tasks. Recurrent neural networks (RNNs) belong to the class of deep-learning methods, and are a popular approach to time-sensitive tasks, as they can back-propagate data representations of previous actions to current actions. This can be useful in identifying dangerous sequences, where the current action influences following actions [13]. As opposed to other RNN variations, LSTMs [21] implement a long term memory architecture, by remembering and forgetting useful information in an internal cell state, and forwarding and selecting from that information in later time frames. This approach works on many longer time lagged problems. As a consequence, LSTMs are applied to many sequential classification problems, like speech recognition [9], and may also fit the problem of identifying dangerous sequences.

In identifying dangerous sequences, we encounter unequal class distribution, which is a well-known problem in the field. In the work of Leeftink [22], about 5% of the sequences was considered dangerous. Fortunately, the field of machine learning already has various methods to increase performance on unbalanced datasets. Rebalancing belongs to the potential solutions, which can be done using Imbalanced-learn [23] when using Python. The three most common ways are reducing the number of samples from the majority class (undersampling), sampling the minority class more often (oversampling), or synthetically generating more samples. Generating more samples may introduce confusing examples in high-dimensional problems, so synthetic generation is not desirable in our example. Undersampling and oversampling

make the dataset more balanced by changing how often positives and negatives occur. If too much oversampling is performed, the model may focus too much on examples that in reality are not informative, which causes overfitting. If too much undersampling is performed, it is possible that important types of sequences are under considered, or so many sequences are removed that results become unreliable. An alternative to rebalancing is boosting [24], like AdaBoost, where the model consists of multiple layers. In each layer, the wrongly predicted examples of the previous layer carry more weight or are sampled more often, which naturally focuses on correctly predicting examples the wrong examples from the previous layer. This way, you naturally bend your model towards decreasing errors on difficult examples, which can be a whole class in imbalanced datasets. Both boosting and rebalancing are examined as possible methods to tackle imbalance in our dataset.

2.3 Identifying Dangerous Sequences using LSTMs

In the area of soccer, RNNs and LSTMs are already used for action classification from soccer video fragments [10]. Verpalen [11] tried to use raw coordinates as input to LSTMs, in order to predict future player movement, and showed that lower sample intervals improved performance. Although this type of input and model is close to the type and model used in our thesis, the problem of identifying dangerous sequences is generally approached using pitch-based input. Using different types of neural networks, Fernandez et al. tried to find high-value regions of the field, based on the assumption that over multiple games, players will generally concentrate around high value areas, even when the positioning is deviating between situations, due to factors like opponent positioning [5]. They used this to model action likelihood using a convolutional neural network on top of pitch control and pitch influence surfaces to better visualise important areas on the pitch [18]. Closer related to this thesis is the work by Dick and Brefeld [25], who propose deep reinforcement learning using a multi-channel pitch map to rate game situations according to their potential to lead to successful attacks, based on reaching the last 18 meters of the pitch. Dick and Brefeld al. point out that their methods are data-driven, as opposed to the method of Link et al., who use an aggregate, dangerousity, which depends on expert knowledge. The works by Dick and Brefeld and Fernandez et al. both use a multi-layered pitch representation to capture all context of the field. For our work, this approach will not be suitable, as we want to take advantage of the many single value features already available, and these do not scale well to a multi-layered pitch representation.

Closest related to this thesis is the unpublished work by Leeftink, in which he uses LSTMs to classify soccer sequences using the danger success definition [22]. His results were sometimes inconclusive and unstable due to the usage of datasets with only 4 or 35 matches. We have the same goal. We could confirm the methods of his work by performing the experiments on a larger dataset. Additionally, we want to give insight into which of the available features influence the prediction of dangerous sequences the most, including features that were previously almost unusable due to the many NaNs they naturally contain.

3 Data

In this section, we will discuss the data used during our experiments. We have access to a dataset with 302 matches of the highest professional soccer competition in the Netherlands,

the Eredivisie. For the 302 matches, we have both event data and tracking data. These are combined into sequences, using the methods described in Section 4.1.

3.1 Event Data

Event data consists of discrete, mostly on-the-ball actions, and the teams and players involved in these actions. Actions can be enriched with important event specific contextual information. This context can be, for example, whether a shot missed the goal, if it was saved by the keeper, or which body part was used to shoot. It is generally a high level type of data, which means the data is structured to common terminology to make it easy to interpret.

In this thesis, events are provided with timestamps, rounded to fit in one of the 10 frames per second. We use two types of events: *ballwin* events and *danger* events. A ballwin event roughly means a team gained possession of the ball, and kept it for at least two actions. They cannot be solely headers or deflections by the goalkeeper. Ballwin events can be one action, if that one actions is a danger events or the keeper catching the ball. A ballwin event stops when the opponents gain possession. *Danger* events indicate that a dangerous event has occurred, which include shots, save attempts or goals. These events capture the time intervals of our sequences, using the methods described in Section 4.1.1.

3.2 Tracking Data

Tracking data is used to capture the lower level context at any moment during a match. It contains the position of every player and the ball over time, which can be used to quantify larger game structures, like spatial domination or player lines. In this thesis, the tracking data is also presented in 10 frames per second, and captured using cameras around the pitch. The tracking data provides the input, called features, for our LSTM and baseline methods.

A match with pure playing time would take at least 54000 frames (10 frames x 90 minutes x 60 seconds), which means our data should contain 54000 values for every feature. In reality, features tend to miss large quantities of values, especially when they contain information related to when a player is in possession of the ball. In that case, it is not unusual that features miss around half of their values. Generally, features can still be used by removing sequences with missing data, if not too many values are missing. So, we consider features sparse if at most 15% of their values are missing. The non-sparse features can be found in Table 1, and the sparse features can be found in Table 2.

3.3 Feature Sets

In this thesis, different experiment setups are compared. These setups contain varying parameter settings, where the most important setting is the set of features. In total, we can use 167 features, but using them simultaneously can negatively impact performance. It becomes more difficult and time-consuming to learn the correct patterns that identify when the actions in a sequence create dangerous situations. Because methods can be effective using certain groups of features, we want to evaluate our methods using different feature sets. Also, because it is time-consuming to measure every possible subset of features, we created multiple feature sets. Feature set *Full* contains all features available. Feature set *Base* contains the feature set used

Table 1: The non-sparse features available for the dataset. Non-sparse indicates that the data can be computed for most time frames in the dataset. The features above the dashed line are used in feature set Base, as described in Section 3.3. When possible, features have variations: width (X-coordinate) or length (Y-coordinate)*, available for both teams**, or both ***.

Non-sparse Feature	Description
TeamCent***	The team centroid, roughly the average player position.
Length**	Distance between the player of this team closest to own goal and the player closest to the other goal.
Width**	Distance between the player of this team closest to the left side line and player closest to the right side line.
Spread**	Standard deviation of distance to the centre of the team.
stdSpread**	Standard deviation in team spread.
Surface**	Surface covered by the teams.
SumVertices**	Circumference of surface area of the teams.
ShapeRatio**	Ratio between team width and team length.
Ball*	Position of the ball.
smallestDistToBall	Distance to the ball from the player closest to the ball.
is_inPossession	Any player has the ball (boolean).
ballDistance	Distance covered by the ball (in the last second).
ballDirection	Orientation of ball displacement (radius in the last second).
ballDirectionChange	Orientation of ball change (radius in the last second).
ballDirectionChangeWeighted	OrientationChange * Distance of displacement (radius in the last second).
ballCarrierInside16m	True if a player has the ball and is within 16 meters of the goal (boolean).

in previous work on this framework [22], and can be found in Table 1 as the features above the dashed line. This set contains features with few NaNs.

3.4 Sequences

In total, we can retrieve 51494 sequences from the 302 matches Eredivisie dataset. The competition contains 18 teams, which means that 306 matches are played, and four matches are missing from the dataset. Unfortunately, some features are missing for a subset of matches, so we can use 266 matches with 45301 sequences, of which approximately 26362 have a valid interval length (with a duration of at least 10 and at most 40 seconds) and are correctly formatted. Valid interval length is further explained in Section 4.1.5, which also explains why we consider both the first ten seconds of a ballwin event and the last ten seconds of a ballwin event as input for our models. Further sequence information largely depends on the experiment setup. In the experiment setup of feature set Full with NaN-filling and no sampling, we have 26304 sequences of which 2691 are positive, 10.2% of the data is labelled positive, and each match contains approximately 99 valid ballwin events. Without NaN-filling, we have only 2988 sequences, of which 222 are positive. For Feature set Base without sampling, we have 26362 sequences with 2695 positives, and 26355 sequences with the same number of positives without NaN-filling.

Table 2: The sparse features available for the dataset. Sparse indicates that the data cannot be computed for many time frames in the dataset, and additional processing is required. When possible, features have variations: width (X-coordinate) or length (Y-coordinate)*, available for both teams**, or both ***.

Sparse Feature	Description
UseOfSpace***	Team coverage of certain areas on the field, based on spatial dominance.
ballCarrier*	Position of the player in possession of the ball.
ballCarrier_distToGoal	Position of the player in possession of the ball, relative to the goal.
ballCarrier_distToGoalLine	Position of the player in possession of the ball, relative to the goalline.
lastDefender*	Position of the last defender of the defending team.
lastDefender_distToGoal	Distance between the last defender of the defending team and his goal.
euclidDistToLastDefender	Euclidean distance of ball carrier to last defender.
relDistToLastDefender	Axis distance of ball carrier to last defender.
closestDefenderToBallCarrier*	Location of the defender closest to the ball carrier.
dist_closestDefenderToBallCarrier	Distance between ball carrier and closest defender.
Link_Zone_TeamScore**	The danger of a goal being scored from the position of the player on the ball.
Link_Control_TeamScore**	The extent to which the player can implement his tactical intention on the basis of the ball dynamics.
Link_Pressure_TeamScore**	The possibility that the defending team prevents the player from completing an action with the ball.
Link_Density_TeamScore**	The chance of being able to defend the ball after the action.
Link_dangerousity_TeamScore**	Aggregate of the four Link features above.

In Figure 1, we see that ballwin events are most likely short, where the chance of a specific ballwin event duration occurring gradually decreases when the total duration increases. Where danger events occur during ballwin events can be found in Figure 2. This figure shows similar patterns in when danger moments occur in ballwin events. Together, they show that danger events are often found at the end of a ball possession, as explained in Experiment 5.2.

4 Methods

Our main focus in dangerous sequence classification is the LSTM with AdaBoost. First, we describe how we prepared the data. Then, we will present the LSTM and the four baseline methods used to identify dangerous sequences. Lastly, the metrics used to compare our classification methods are described.

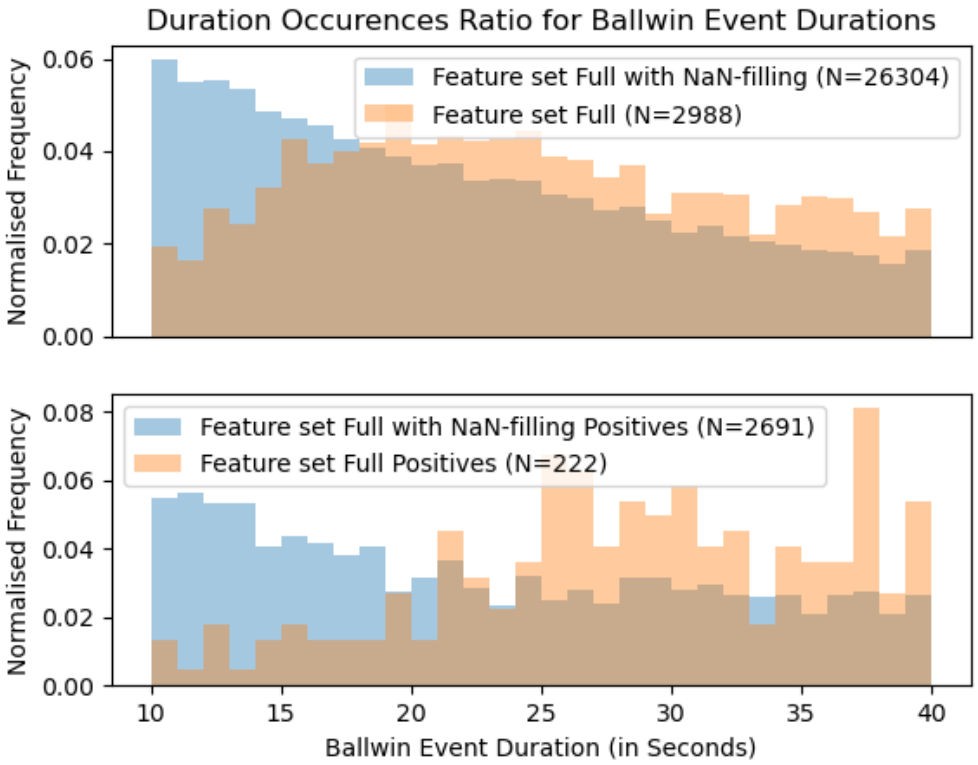


Figure 1: The number of occurrences for the ballwin event durations before data preparations, for feature set Full, without sampling, both when using NaN-filling and when not using NaN-filling. The bottom graph shows the events that are labelled positive for both the feature sets in the top graph.

4.1 Data Preparations

In this Section, we discuss the practical challenges in using our event data and tracking data, and combining these into the sequences of our classification methods.

4.1.1 Identifying and Labelling Sequences

The event data of our dataset is used to find the start and end of sequences. We do this by searching for ballwin events, and taking the corresponding time interval. Then, we label time intervals as successful or not successful, based on whether a danger event can be matched to the ballwin event.

A danger event and ball possession can be slightly out of sync, which is mostly due to one (or more) of three reasons: small deviations, because the danger events are manually registered, some ball possessions do not qualify as ballwin events, or it may be difficult to clearly identify when teams are in possession of the ball. So, we match danger events and ballwin events on certain conditions.

1. If a ball possession is not qualified as ballwin event, it is not meaningful enough, and another ball possession is assumed to contain the context we require.

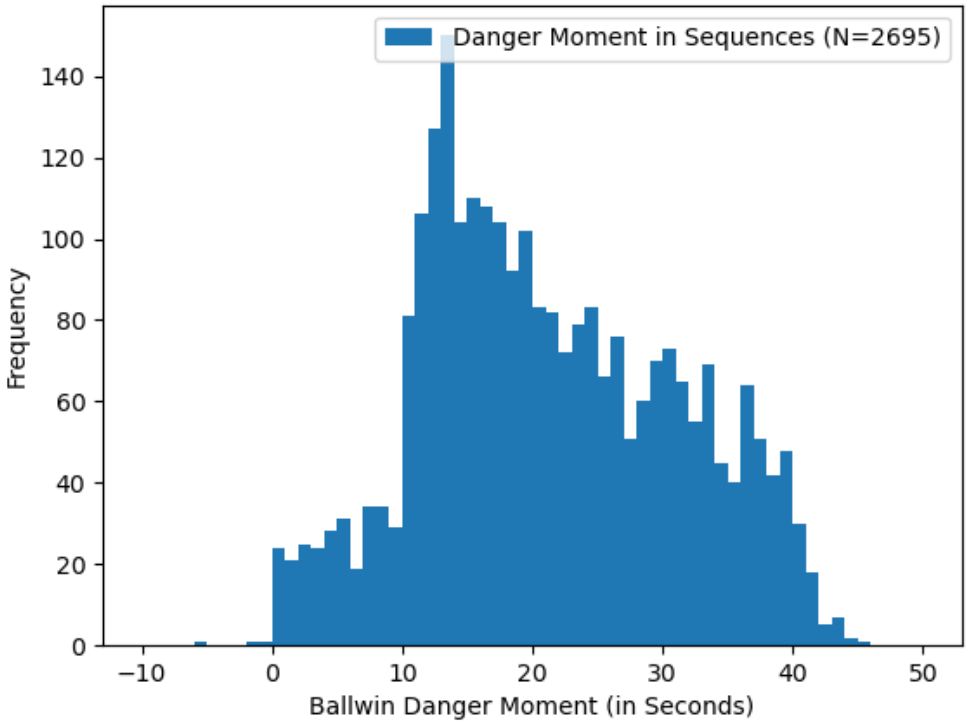


Figure 2: The moment where the danger event occurred in dangerous ballwin events, and the frequency of that timestamp.

2. A danger event is registered either in a ballwin event, or at most six seconds before or after a ballwin event, inside the leeway of a sequence (indicated by arrow two, five and seven in Figure 3).

3. We require the danger and the ballwin events to be from the same team.

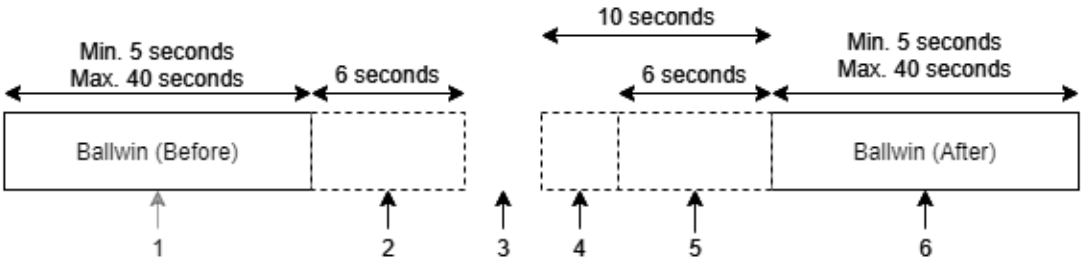


Figure 3: Matching danger events and ballwin events. Arrows two to six indicate possible danger event occurrences. Arrow one does not exist, as in ballwin after will be ballwin before in this scenario. The time frames presented by arrows two to five can overlap.

If these conditions fail, a danger event and ballwin event are not matched. Now, for every danger event, we obtain the last and the next ballwin event. If the assumptions hold for one ballwin event, it is matched to the danger event. It is possible that both ballwin events qualify. This means that the time frames belonging to arrow two to five are overlapping. In this case,

we prefer the after ballwin event, but only if the danger event happened at most 10 seconds before that event (indicated by arrow four and five in Figure 3).

4.1.2 Handling NaNs

Our framework contains a large library of existing features, of which most features are computed for every frame in the Eredivisie dataset. Unfortunately, feature data corresponding to a frame of a sequence can be missing. In previous work on our framework, this meant the whole sequence was removed, as LSTMs by default will not process missing data. The number of removed sequences can become very large, as seen in Figure 1 as the decrease in number of available sequences. Throwing away sequences means available sequences are more reliable, but having less training sequences can create a bias or reduce overall reliability, and it also means that features that naturally contain more missing (NaN) records were previously almost unusable in the framework.

A common approach to the problem of handling missing data is imputation, but this has two disadvantages. With imputation, we try to fill the empty records with data, generally by estimating the best value to fill the gap with. However, a missing record value on itself can be valuable to models, and filling records with estimated values makes the data less reliable, as the new data is an approximation of what we think should be at the NaNs. We attempt to make the features usable by applying NaN-filling to the tracking data features. We examine two methods: an unrealistic constant value is set, or we perform forward fill, which uses the last known value for a feature in the next frame. Condition on unrealistic values is that the value has no meaning to the feature, and occurs enough to be trained on. It will greatly affect performance if the model fails to recognise the unrealistic value, as the feature can become too corrupted to be informative. Forward fill can smooth nuances which are important for classification, especially when a feature has a lot of consecutive NaNs. Still, when effectively used, NaN-filling can allow us to save many sequences, especially when using NaN-rich features, and the increased number of available sequences can greatly increase model performances.

4.1.3 Sampling

In classification problems, it can negatively affect performance when one class heavily outbalances the other classes. We may overfit the models on the majority class, which makes it difficult to generalise on the minority class. Boosting can be used to decrease dataset imbalance [22], but we also examine sampling methods undersampling and oversampling to decrease dataset imbalance. Sampling rearranges the data to create a more balanced dataset, either by introducing more samples of the minority class (oversampling), or removing samples of the majority class (undersampling).

We implemented the undersampling and oversampling method by Lemaître et al. [23]. These methods are provided a sampling strategy, which is defined as the ratio of the number of samples in the minority class over the number of samples in the majority class after sampling. This ratio can be written as $\mathrm{Ratio} = S_{mi}/S_{ma}$, where S_{mi} gives the sample size of the minority class, and S_{maR} is the sample size of the majority class after resampling. Before using sampling methods, our minority class to majority class ratio is approximately 10.2%, with 10.2 dangerous sequences for every 100 negative examples. In our experiment, we set the desired ratio to 0.25 for either undersampling or oversampling. They are also used together, in which case we first

use 0.25 as undersampling strategy, then 0.4 as oversampling ratio. These values are arbitrary. They are chosen to have a great enough impact on the dataset so that actual improvements to the models can be detected, but decrease in reliability due to removing too many samples is unlikely.

4.1.4 Normalisation

We normalise our playing direction, which means data based on player direction changes in value. Generally, features using X or Y coordinates, like ball, player or centroid position, require normalisation, and distance measures, like distance to opponent goal, do not require normalisation. Because coordinates are measures from the centre spot and projected on a 105 by 64 meter pitch, normalisation on positions generally only requires us to multiply X and Y coordinates by -1. BallDirection is the exception, to which π is added and the result is mapped between zero and 2π. Two examples of feature normalisation can be found in Figure 4. If a sequence contains any missing values or is the wrong length, it will be deleted.

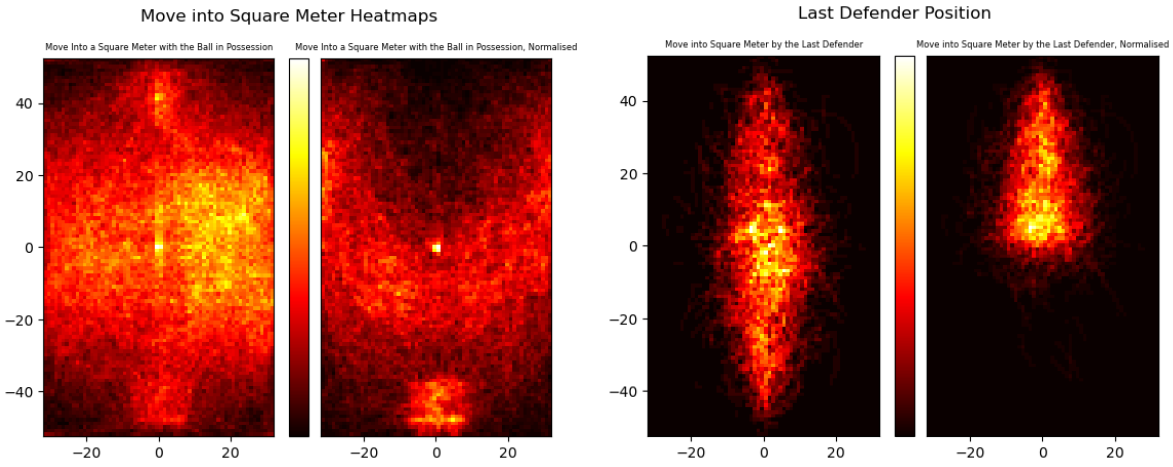


Figure 4: Feature normalisation for last defender position and ball position (when in possession). For both sub figures, left is an non-normalised figure of the features, and right is normalised. Attacks occur from bottom to top. The X and Y axis roughly present field coordinates measures from the centre spot. Coordinates are rounded to square meters. Results are from a 266 match dataset.

4.1.5 Sequence Length

Not all ballwin events are interesting enough to consider. When the ball is captured, and immediately recaptured by the opponents, it will be difficult to state something useful about the short time of possession. Likewise, when a ball possession is very long, it can be difficult to detect important actions during the sequence. Therefore, we set a minimum and maximum duration on a ballwin event. We consider a ballwin event valid when it spans at least a couple of seconds, so we initially set the minimum to 5 seconds, and the maximum to 40 seconds.

Preferably, we would like to use a whole ballwin event. However, having variable lengths of intervals can negatively influence our prediction capabilities. For instance, it is considered easier to score from counters, as the defence is not fully structured in those situations. As counters

generally have a short duration, it is easy to create a bias based on sequence length. Additionally, as mentioned earlier, it is harder to detect important actions in longer intervals. Thus, we also want to select a fixed length time interval within our available time interval. Sequences with a fixed length time interval will be mentioned as a valid sequence.

Eventually, we want to be able to predict danger from intervals of the first ten seconds of a ballwin event, but this might be difficult or even impossible to do if danger depends on actions in the last seconds of a ballwin event. In Section 5, we investigate the possibility to use the last ten seconds of a sequence. Because our selected interval length is ten seconds, we also increase our minimum interval duration to ten seconds. This means that each sequence contains the first ten or the last ten seconds of a ballwin event, depending on the experiment setup.

4.2 Baseline Methods

To evaluate the results of our LSTM with AdaBoost, we compare its performance with four widely accepted baseline methods for machine learning. These are implemented using Scikit-learn [26].

4.2.1 Random Forest Classifier

Random forest classifiers [27] belong to the group of ensemble machine learning methods. In such methods, multiple weak learners are combined into one stronger predictor [24]. In random forest classifiers, we generate a forest of trees, where each forest is created from a random set of features. The prediction is the label that the majority of the trees present.

The advantages of random forest classifiers is that it is often faster to train multiple weak learners than one strong learner. Also, due to majority vote, the risk of overfitting on certain trees or features is reduced. Because we have access to a large library of features, a reduced risk on overfitting is a huge advantage.

4.2.2 Naive Bayes

Naive Bayes belongs to the group of probabilistic classifiers, which apply Bayes' theorem to get the likelihood of obtaining each class. For Naive Bayes, we compute the probability of example $E = (x_1, x_2, ..., x_n)$ being class c, where x_i is a feature value for that example, and n is the total number of feature values. The probability of class c, given feature values E, is written as $p(c|E) = \frac{p(E|c)p(c)}{p(E)}$. We then pick the class c with the highest likelihood. Naive Bayes naively and per definition assumes that all features are independent of each other.

Naive Bayes is fast, requires relatively small datasets, is relatively insensitive to irrelevant features, and it handles missing feature values well. However, a drawback is the assumption that all features are independent of each other, which could mean that more complicated relations of multiple features will not be effectively recognised.

4.2.3 Logistic Regression

Logistic Regression is a type of generalized linear model (GLM). Linear models use function values to separate prediction classes. We try to find the function that best fit data by calculating the error of a function. If the error model is not normal distribution based, the linear model fits in the group of generalised linear models. Logistic regression utilises a sigmoid function.

Logistic Regression is fast, simple, and often used as baseline. It is easy to interpret its results, especially in discovering linear relationships between features. However, it is less successful on non-linear problems or with complex relationships, which is likely in a soccer dataset.

4.2.4 AdaBoost Classifier

AdaBoost Classifiers [28] belong to the group of ensemble machine learning methods and boosting methods. In ensemble methods, multiple weak learners are combined into one stronger predictor [24]. A subgroup of ensemble methods is boosting, where, instead of a majority vote from the collection of weak learners, weak learners and data examples are iteratively weighted. In AdaBoost Classifier, we take the results from previous weak learners, and focus more on the examples that were previously misclassified, by increasing their weights in the next weak learner. The total weight of each weak learner is based on the number of correct classifications. The weak learner used in AdaBoost are single split decision trees, called decision stumps.

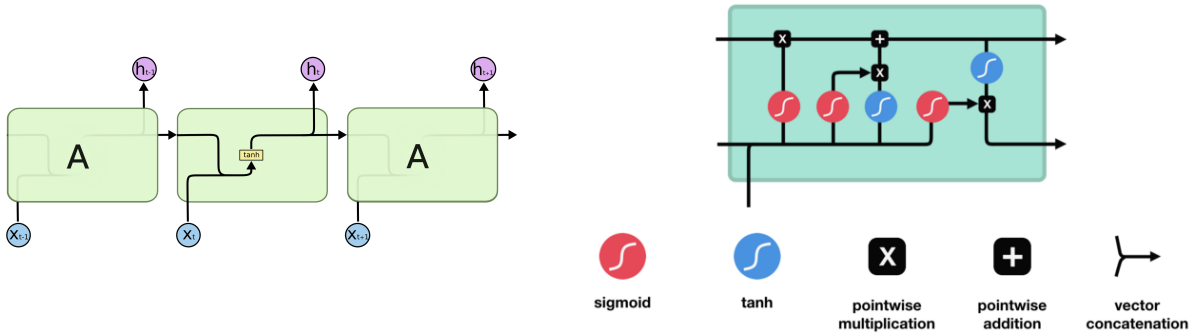
AdaBoost works well on imbalanced datasets. Generally, the tendency in imbalanced datasets is to misclassify examples of the minority class, but AdaBoost focuses on those misclassified examples and, consequently, the minority class. However, AdaBoost is sensitive to noisy data and outliers, which makes it ineffective in various practical situations.

4.3 LSTM with AdaBoost

The main focus in this thesis is the boosted LSTM. We first describe the LSTM itself, followed by how it is boosted using AdaBoost.

4.3.1 LSTM

Long Short-Term Memory [21] networks (LSTMs) belong to the group of Recurrent Neural Networks (RNN). Recurrent Neural Networks are Neural Networks that use the concept of time space, by feeding back representations of previous events into the current time step. LSTMs became a popular approach to sequential data, because they were successful in introducing long term memory to problems with larger time lags, and outperformed competing algorithms in complex real world problems, like in speech recognition [9]. Competing algorithms generally relied on short-term memory applications, for instance by delaying time steps. However, such solutions did not perform well on larger time lags problems. Because the model error is generally back-propagated through a network of layers, errors could explode or vanish before enough time passed to bridge long time lag. As a results, competing algorithms often take long to learn the problem, or fail completely. LSTMs managed to perform stable on noisy input in lag intervals of a few hundred time steps, without suffering loss on short time lag problems, which meant that the error flow did not explode or vanish.



(a) A series of RNN cells. At time step t, X indicates the input vector and h indicates the hidden state vector. LSTM networks follow the same structure, but have more advanced cells.

(b) Memory cells in an LSTM. Gates can be recognised as sigmoid functions, followed by pointwise multiplication. The upper stream contains the cell state, the lower stream contains the hidden state, concatenated with the current input, and eventually turned into the output data.

Figure 5: Source RNN cells [29] and LSTM cell [30].

LSTMs consist of subsequent RNN cells (Figure 5a), expanded with the concept of gating. Gating is the method of using pointwise multiplications to multiply input with a vector of values between zero to one. Like valves, this allows certain percentage of the input to flow through, essentially selecting the information to pass, where closer to zero means more data is forgotten. To select which information we pass, we train neural networks, and to guarantee a value between zero and one, we use a logistic sigmoid function.

In LSTM cells (Figure 5b), we have a cell state c_{t-1} that holds our long term memory, and we have the concatenation of current time step input data x_t and previous hidden state h_{t-1}, which we call and $x_t + h_{t-1}$. The hidden state is the output data of the previous cell. In a cell, we encounter several phases:

1. The forget gate uses $x_t + h_{t-1}$ to decide which data from the cell state to forget, and which data to remember in its cell state.

2. Similar to RNN cells, we regularise our $x_t + h_{t-1}$ data using a than-function, that transforms that data to a scale from -1 to 1. In LSTMs, however, we use a input gate to decide what data is relevant to remember. This data is pointwise added to the cell state, resulting in the new cell state.

3. After regularisation, the cell state now flows into the output gate, where the unregularised $x_t + h_{t-1}$ data decides which cell state data to use. We now have our next hidden state. The last hidden state will give our output data.

In our implementation of the LSTM model, we use a binary cross-entropy loss function with Adam optimisation to minimise the error of our prediction model, as these are ordinarily used for prediction models in similar problems. Our LSTM model consists of two 32-node LSTM layers. This is followed by a fully connected layer of 16 nodes with a relu activation function and an output with a sigmoid activation function. The architecture of our model is described in previous work [22].

4.3.2 Boosted LSTM

In Section 4.2.4, we explained how a standalone AdaBoost Classifier works. We combine multiple learners, called the estimators, into one better learner, with the learners being decision stumps. When boosting a model, we replace the type of learners, to LSTMs in our case. We train the models iteratively, where the wrongly predicted examples of the previous iterations have more weight in the next iteration. However, each iteration also has a reduced weight compared to the previous iteration, which allows us to generalise in the first layer, and specialise more in each consecutive layer.

It should be considered that for each estimator, a model has to be trained. This could dramatically increase time, so this method should be used with caution.

4.4 Metrics

To evaluate our classifying methods, we use a set of metrics. Each metrics has advantages and disadvantages, and the set of different metrics gives a better view on the performance of our methods. These metrics are implemented using Scikit-learn [26].

Classification tasks using two possible output classes are called binary classification tasks. In binary classification, the label used by the metrics are called positive (successful) and negative (unsuccessful), and the prediction is further described by whether the prediction was true or false. The four possible categories are true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). The metrics score the success of a classification method based on their performance in certain classes.

4.4.1 Accuracy

A simple way to compute performance of a method is by measuring the accuracy. Accuracy is the number correct prediction over the total number of predictions. Accuracy gives a score between 0 and 1, where 1 means that all examples are correctly predicted, and 0 means that all examples are wrongly predicted.

$$\text{Accuracy} = \frac{\#\text{ Correct Predictions}}{\#\text{ Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

The measure is easy to interpret, but gives no information on classification success in each class. This is especially problematic for imbalanced datasets, where simply predicting the majority class in binary classification would always result in a high performance indication.

4.4.2 Precision and Recall

Precision and Recall are two similar metrics that are often confused. Precision focuses on the quality of your predictions, and checks how many of positive predictions were correct. Recall focuses on how well a class was predicted, and checks the amount of times the actual positives were predicted correctly. Precision is used to see how well predicting potential positives performs as the likelihood of the classifier to be correct, where recall shows how well your class of positives are predicted as whether your model will detect the positive examples. Precision

and Recall gives a score between 0 and 1, where 1 is the best possible score, and 0 is the worst possible score.

$$\text{Precision} = \frac{\#\text{ positives correct}}{\#\text{ labelled positive}} = \frac{\text{TP}}{TP + FP}, \text{Recall} = \frac{\#\text{ positives correct}}{\#\text{ actually positive}} = \frac{TP}{TP + FN}$$

Precision and Recall are useful in examining the positive predictions and the positive examples, but are not informative in investigating the overall performance of both classes. They are asymmetric, which is definitely a limitation when working with imbalanced datasets. Despite the limitations, it can be insightful to know how we classify the positive examples we have using Recall and, probably to a lesser extend, how many examples are classified as positive. However, we will not be using these stand-alone metrics, as a combination of the two exists in the F1 score.

4.4.3 Area under the ROC Curve

In a binary classification problem, we can set a probability threshold between 0 and 1, where each example with a probability above the threshold is predicted to have a positive label, and examples below the threshold are predicted to be negative examples. By lowering the threshold, we will encounter more positive examples, but also an increased number of wrongly classified negative examples. The True Positive Rate (TPR) and True Negative Rate (TNR, or Recall) are used to present the number of examples belonging to each class, and these are defined as:

$$TPR = \frac{TP}{TP + FN}, TNR = \frac{TN}{TN + FP}.$$

By computing these rates at certain intervals, we get a curve that we call a Receiver Operating Characteristic (ROC) Curve. If a model makes a clear distinction between the two classes, the difference between positive and negative examples would be larger, and we would introduce relatively fewer negative examples when lowering the prediction threshold. So, a better performing model would generally have a higher TPR and TNR at more intervals, which is measured by a single number between 0 and 1 as the Area Under the Curve (AUC). By using TPR and TNR directly, our information is asymmetric and susceptible to unbalanced datasets.

4.4.4 F1 Score

F Score combines Precision and Recall into one measure that simultaneously investigates the performance of predicting positives and how well the positive class performs. F1 indicates that both Precision and Recall are weighted equally. F1 Score punishes the Precision or Recall when one is much higher than the other. For a model, this gives a better perspective on both the capabilities to detect potential positive examples, as how precise it classifies them. F1 Score outputs a score between 0 and 1, where 1 indicates a perfect Precision and Recall prediction, and 0 means the worst Precision and Recall prediction.

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Because F1 Score balances both Precision and Recall, the combination is generally effective on imbalanced datasets. Unfortunately, it is still focused exclusively on the positive examples, which means it is not as useful in every experiment setup.

4.4.5 Average Precision

We can substitute the TPR and TNR of a ROC curve by Precision and Recall, and we get a Precision-Recall (PR) Curve. The AUC of the PR-Curve then roughly translates to Average Precision (AP). By using Precision and Recall, our information is still asymmetric, but less susceptible for data imbalance due to independence of the actual number of examples in each class.

4.4.6 Matthews Correlation Coefficient

Matthews Correlation Coefficient [31], or MCC, takes into account the all classification categories, while reducing the limitations of category size. For MCC, the best possible score is 1, indicating a perfect correlation between the predictions and actual labels, and the worst possible score is -1, indicating a perfect negative correlation between the actual labels and predictions.

$$\mathrm{MCC} = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Matthews Correlation Coefficient involves both the positive and the negative class in its calculations. Because it is symmetric, it gives a better view on performance over all classes. Because it takes all possible outcome categories into account, it also performs well on imbalanced datasets.

5 Experiments

In this thesis, we expanded on the machine learning section of our framework. The goal of this thesis was to improve existing boosted LSTMs to (better) classify sequences using the *danger* definition, or at least better understand its potential. Our contributions are evaluated using a series of experiments, using a non-synthetic, full-season Eredivisie dataset.

Experiment I measures the performance gain by introducing new features to existing models. If sequences contain NaNs, which LSTMs by default cannot process, these sequences are removed from the dataset. However, this leads to deletion of most sequences when the new features are added, as some features contain many NaNs. Experiment II measures the validity of the new dataset. Experiment III measures the performance gain by introducing NaN-filling. It allows the introduction of features that naturally contain large amounts of NaNs, without removing sequences containing them. This should allow an LSTM to recognise and omit missing data. Experiment IV describes how we perform sampling as a possible method to counter dataset imbalance. Additionally, sampling can be used to resize datasets without altering the data itself, which can be useful to get more or better training from small datasets, like the dataset with Feature set Full without NaN-filling.

5.1 Experiment I - Additional Features

In this experiment, we examine the effect of adding a new set of features (of which some NaN-rich) to our existing framework. Because the effectiveness of a feature set can depend

on the parameter settings of models, we measure performance of different experimental setups.

Each setup is done using both the first and the last ten seconds of a ballwin event (as explained in Section 4.1.5), with one, five and ten estimator LSTM boosting, and using two different feature sets. We examine the performance of baseline methods, to validate that performance patterns are not limited to LSTMs. Feature set Base exclusively contains features without natural NaNs, which allows us to better identify the influence of small amounts of NaN-filling (in later experiments) and small amounts of additional sequences. Feature set Full contains all available features, including some NaN-heavy examples. This experiment shows the performance gain by introducing new (NaN-rich) features, which naturally results in having a smaller amount of training data, as we delete sequences containing any NaNs.

5.1.1 Results - LSTMs

Figure 6 and Figure 7 present performance of different parameter setups for LSTMs, focusing especially on using one, five or ten boosting estimators. In this experiment, we ignore feature sets with NaN-filling. Using LSTMs with five estimators outperforms LSTMs with ten estimators in most setups. Both perform better than the LSTMs without boosting. It is not clear if using five estimators outperforms using ten estimators when using the first ten seconds of ballwin events for feature set Full. Results for feature set Full are inconsistent and fluctuate, as can be seen in the top two lines or the bottom three lines in either Figure.

Average precision seems more constant and higher when using the last 10 seconds of ballwin event, for any number of estimators. Average precision is increasing over time for feature set Base. The slope is increasing from the start of the first 10 seconds, and this trend continues into the last 10 seconds. All methods outperform a model that simply guesses based on the ratio of positives in the dataset.

5.1.2 Results - Baseline Methods

For the baseline methods, results can be found in Figure 8 for using the first 10 seconds of a ballwin event, and in Figure 9 for using the last 10 seconds. Again, we only examine feature sets without NaN-filling. In line with LSTM results, average precision is higher when using the last 10 seconds of ballwin events, and we find a trend of increasing average precision for feature set Base. Also, average precision is inconsistent for feature set Full, but it generally performs better than feature set Base.

The best performance is achieved by Logistic Regression using the Full feature set. AdaBoost Classifier, Random Forrest Classifier and Logistic Regression all perform best in a subset of the experiment setups. Naive Bayes performs worst in most setups.

5.1.3 Results - LSTMs vs Baseline Methods

Baseline methods achieve similar or, more commonly, higher average precision in most equal setups. The highest average precision is achieved by using feature set Full. Using the first ten seconds of ballwin events, boosted LSTMs and Logistic Regression achieve the best results. In predicting danger for the last ten seconds of an ballwin event, Logistic Regression achieves

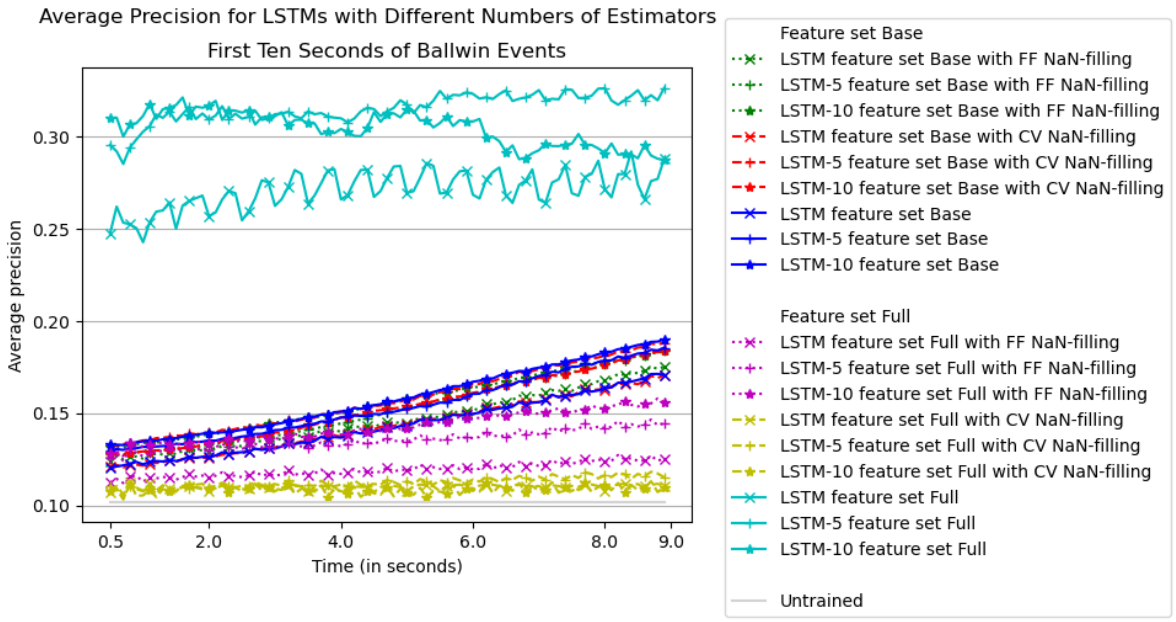


Figure 6: Average precision over boosted LSTMs with different amounts of estimators. The first 10 seconds of ballwin events from 266 matches are used, with and without NaN-filling, for feature set Base and Full.

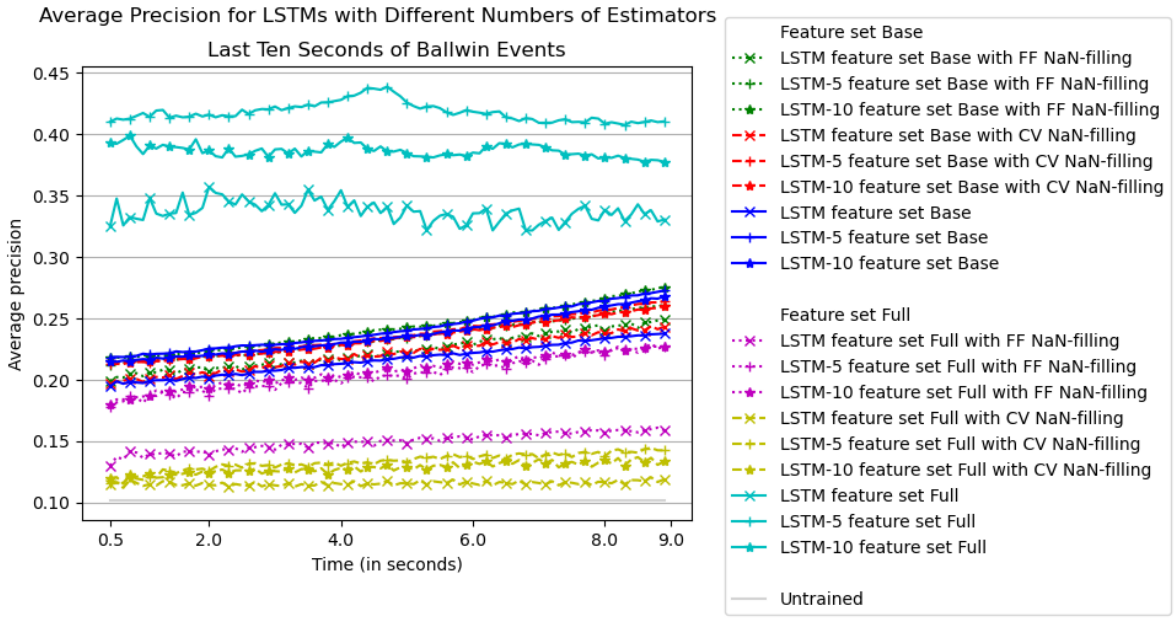


Figure 7: Average precision over boosted LSTMs with different amounts of estimators. The last 10 seconds of ballwin events from 266 matches are used, with and without NaN-filling, for feature set Base and Full.

the highest average precision. Using feature set Base, LSTMs generally perform worse than the baseline methods.

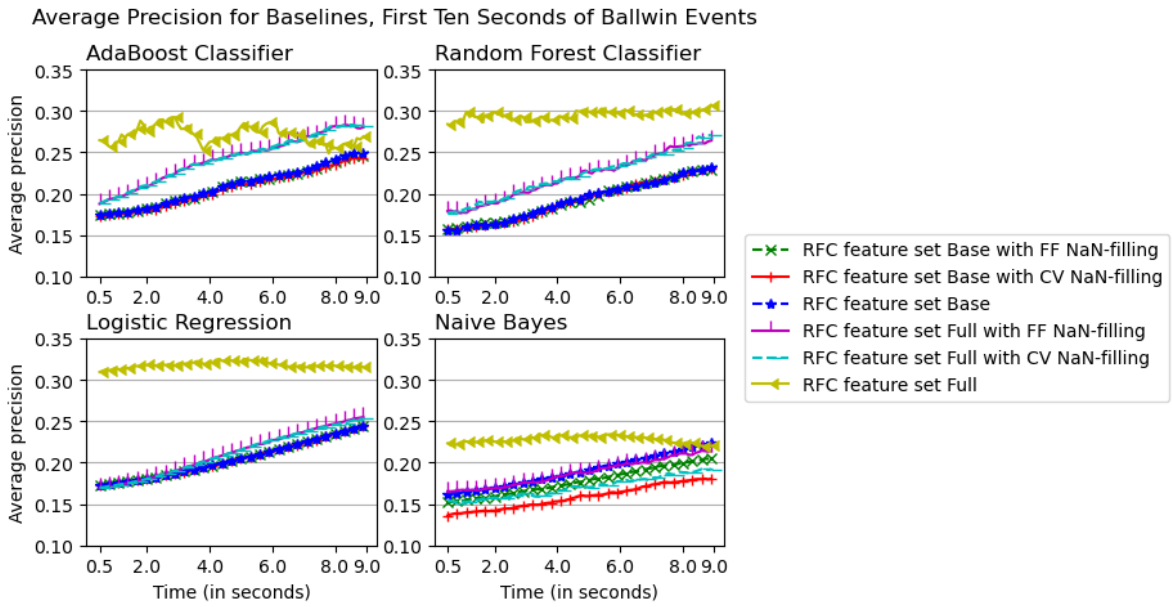


Figure 8: Average precision over baseline methods. The first 10 seconds of ballwin events from 266 matches are used, with and without NaN-filling, for feature set Base and Full.

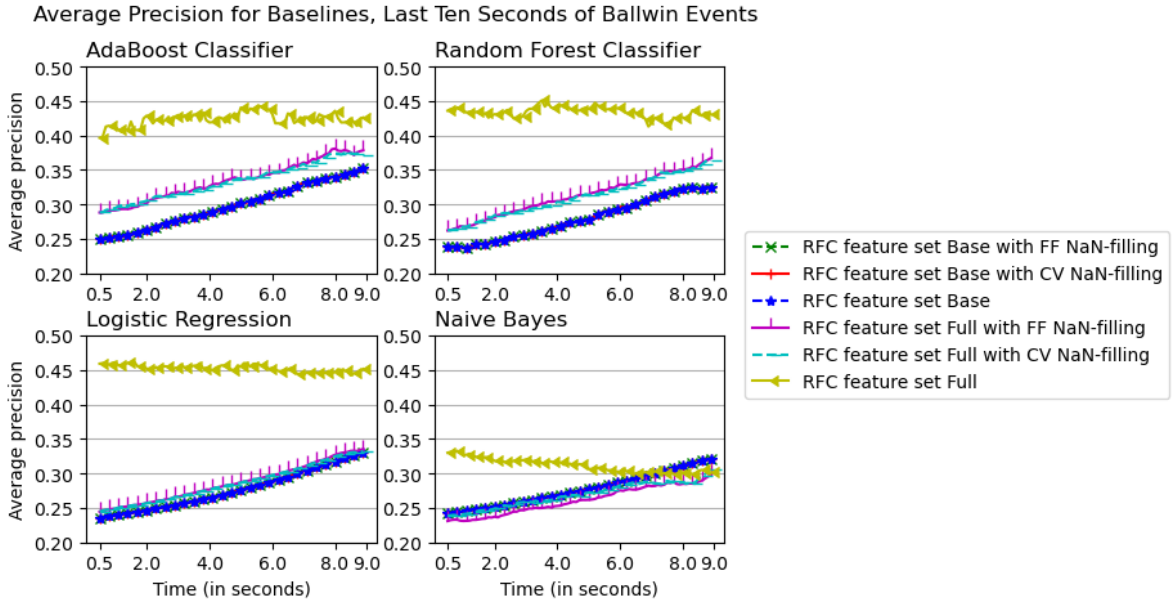


Figure 9: Average precision over baseline methods. The last 10 seconds of ballwin events from 266 matches are used, with and without NaN-filling, for feature set Base and Full.

5.1.4 Discussion

The best average precision was achieved by Logistic Regression, using feature set Full. When using the first ten seconds of ballwin events, LSTMs with five estimators would match Logistic Regression in performance. The baseline methods outperforming LSTMs when predicting danger using the last seconds of ballwin events, but not outperforming LSTMs when using the beginning of ballwin events, possibly shows why the LSTMs perform worse than the baseline methods. It is likely that both the LSTMs and the baseline methods lean on a simple

collection of features to make their predictions, when they fail to comprehend the more complicated structures in the sequences. Logistic Regression performs well on when there are few dependency relations between features. When it becomes more difficult to find patterns from the data, by using the first seconds of a ballwin event for instance, LSTMs, especially with boosting, could increase performance towards the baseline models.

The inconsistency in the average precision for feature set Full shows that it is presumably too difficult to retrieve important patterns from the available data, which only contains 2988 sequences. For models of both the baseline methods and the LSTMs, results fluctuate. This indicates that we did not yet converge on important patterns, which is generally a side-effect of underfitting. If more data was available, the LSTMs and Logistic Regression might not have been the better methods in these experiment setups. This shows the importance of saving sequences or removing NaN-rich features, and is a reason for investigating NaN-filling when introducing NaN-rich features.

The experiment setups that achieved the highest average precision, for both LSTMs as baseline methods, was feature set Full. This setup removed many sequences from the dataset. Whether the dataset Full with removed sequences remained representative for the original dataset is investigated in Section 5.2. If this is not the case, the results can be unreliable, as we could have deleted easier or difficult examples from the dataset, and impact performance by doing so.

We expected the increasing average precision trend, which continues from the first ten seconds of a ballwin event into the last ten seconds. If a sequence contains a danger moment, it is often later in a sequence, as seen in Figure 2, as actions on goal are actions with high risk of losing possession. If it is easier to predict danger closer to the actual danger moment, the problem of identifying dangerous sequences would become easier at the end of a ballwin event. Alternatively, it may be possible that the access to a larger time interval of data may increase performance, which would at least show that our model identifies information over multiple time-steps successfully. This is not likely, as our baseline methods show the same patterns. The trend of increasing average precision does not appear in feature set Full, but we expect this to be the result of overfitting in that feature set.

In our experiment setups, in general, using five estimators seems to be better than not boosting or boosting with ten estimators. This appears to be in line with the results from previous work [22]. This does not hold for Feature set Full, but this is likely due to the possible unreliability of underfitting using this dataset. It is likely that an LSTM would overfit on the data when so many layers are available. This would explain why boosting with five estimators outperforms boosting with ten estimators.

5.2 Experiment II - Dataset Full Representativeness

If we remove as many sequences as in Section 5.1, we have to consider whether the new dataset still represents the original dataset. It is likely that by removing sequences, certain types of attacks are removed from the data more often. If these types of attacks are more difficult or simpler to predict, this could impact model performance. Because grouping similar sequences is a problem on its own, as described in Section 3.1, we limit ourselves to investigating whether

the distribution of sequence lengths remained intact, and if the distribution of positives over these sequence lengths remained intact for the experiment described in Section 5.1.

We also examined the effect of using the sequences of feature set Full in feature set Base. This comparison cannot measure the performance gain of feature set Base by training on more sequences, but measures, more objectively, the performance gain by using a higher quantity of features in feature set Full. Lastly, we measure the performance of feature set Full without NaN-filling using alternative performance measures. This verifies whether average precision is projecting performance for this model realistically.

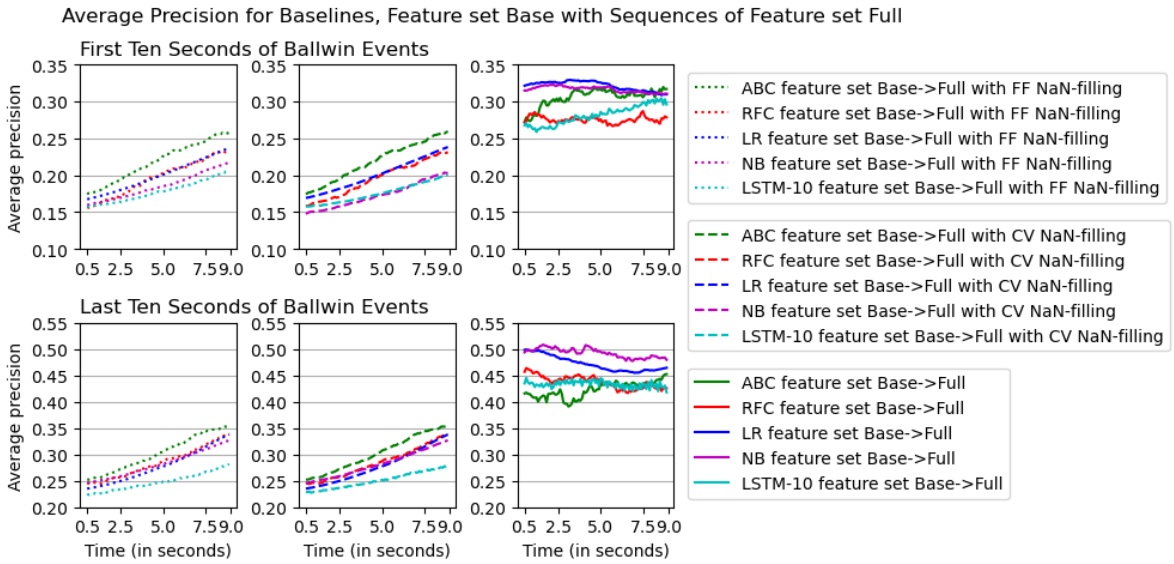


Figure 10: Average precision over baseline methods for feature set Base, using only sequences available in featureset Full without NaN-filling. The first ten or the last ten seconds of ballwin events from 266 matches are used, with and without NaN-filling.

5.2.1 Results

Distribution of sequence length and the corresponding number of positives examples are found in Section 4.1.5. Whether the distribution of positives remained intact is difficult due to the low number of examples remaining in the new dataset, and the expected noise from removing invalid sequences. The distributions of positives in the datasets with and without NaN-filling may correlate, but this is too difficult to observe, due to the low availability of positives in the dataset without NaN-filling. We can see that the sequence length frequency normalisation is decreasing for the dataset with NaN-filling. For the other dataset, patterns are difficult to observe, but we can see the increase in positives for sequences with a length between twenty-two and thirty-two seconds, compared to the succeeding and preceding seconds. It is clear that the distribution of sequence durations has changed. Sequences with a duration between ten and sixteen seconds are underrepresented in the new dataset.

The results of using the feature set Full sequences in feature set Base can be found in Figure 10. In these experiments, using ten estimators generally performed better or similar to using five estimators, which is why the ten estimator results are visualised. In the top and bottom

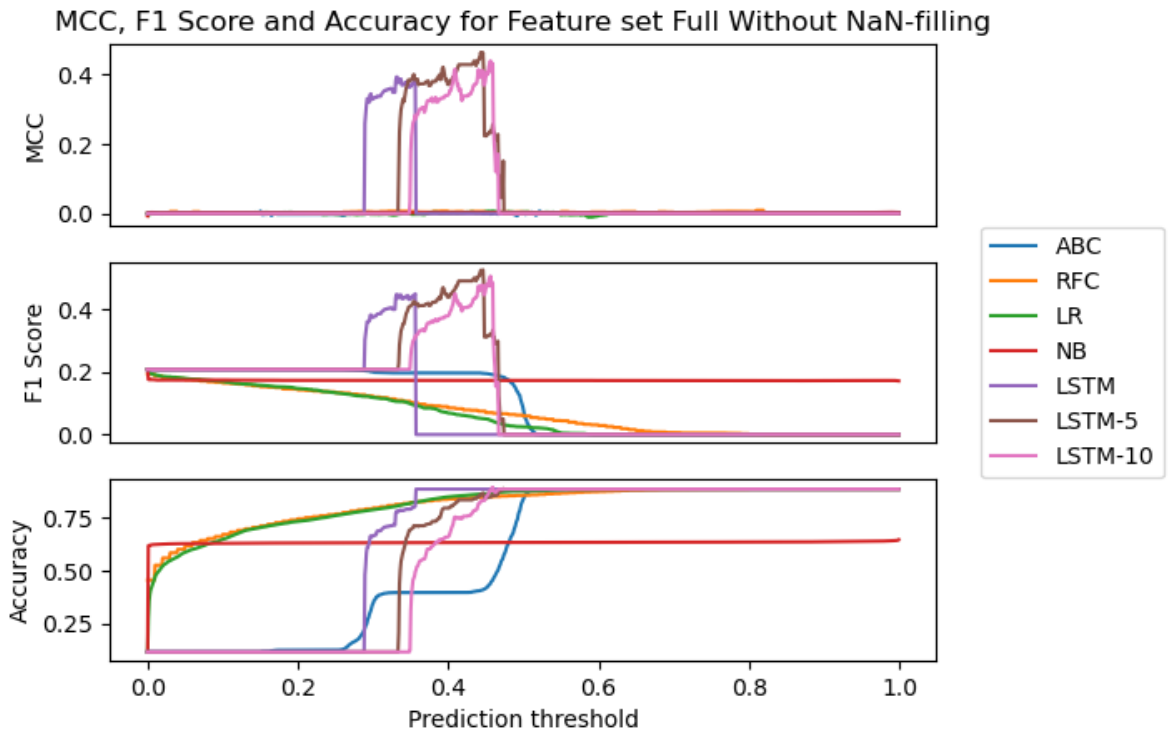


Figure 11: Performance of Feature set Full without NaN-filling for the last 10 seconds of ballwin events from a 266 match dataset.

right figures, we see that results are inconsistent when using the sequences of feature set Full in feature set Base. These results are similar to the best results of feature set Full without NaN-filling, even outperforming that feature set in many setups. Feature set Base using NaN-filling, when using the sequences of feature set Full, performs similar to corresponding setups using all available setups. However, using feature set Full without NaN-filling still outperforms feature set Base with fewer sequences in most setups.

In Figure 11, we see that model Full without NaN-filling, for some prediction threshold, reaches similar performances using other performance measures. Matthews correlation coefficient stays around 0.0 for the baseline methods. Naive Bayes presents flat results overall.

5.2.2 Discussion

To examine the effect of adding more features with feature set Full compared to feature set Base, we isolate the effect of removing sequences with NaNs. This is done by using the same sequences in feature set Full and feature set Base. First, we examine the distribution of sequence lengths, as certain types of ballwin events could become underrepresented or over-represented by removing NaN-rich sequences, which can make predicting danger easier or more difficult. Figure 4.1.5 shows that the distribution of sequences has changed, which is a strong indicator that results will be impacted. This impact is confirmed in Figure 10, as the high but inconsistent average precision achieved by adding the features of feature set Full, at the cost of removing sequences containing NaNs, are similar results to feature set Base when the same set of sequences is removed. Inconsistent performance is often related to underfitting, which is also likely to be the result of the deletion of sequences. This shows that the high average

precision of feature set Full without NaN-filling in Section 5.1, is likely the result of removing large quantities of sequences, and not the result of adding new features.

We find that there is limited difference between using feature set Base with all possible sequences or with the limited set of sequences available in feature set Full. We do see that LSTMs with ten estimators performed slightly better than with five estimators when fewer sequences are available. Possibly, the smaller set of sequences is enough to converge our models on, which results in similar performances. However, a LSTM with more layers was likely better in retrieving more patterns from the limited amount of data. Lastly, we notice that alternative performance measures achieve similar performances to average precision. This shows that the measure average precision is reliable to evaluate the unreliability of feature set Full without NaN-filling.

Using the alternative performance measures, it appears as if average precision is a reliable indicator for feature set Full without NaN-filling, even tough results may fluctuate. This reinforces the thesis that fluctuations are the result of underfitting. Naive Bayes seems to perform relatively constant over the alternative performance measures. This could indicate that it did not properly learn to identify either output class, but always guesses the same class. In that case, we would see a constant line, as it would always make the same predictions. LSTMs appear to perform better using the alternative measures. We did not find a possible reason for the constant results for the baseline methods for Matthews correlation coefficient.

5.3 Experiment III - NaN-Filling

To preserve sequences and have more features available to our framework, in this experiment, we fill empty data with unrealistic constant values (CV), and perform forward fill (FF). We examine whether the introduction of NaN-filling improves model performance. We identify two major influences in the success of our unrealistic value NaN-filling: the model should succeed in recognising the unrealistic value in the framework, and properly recognising unrealistic values can be more difficult based on other model settings, for instance the number of estimators used in boosting or the used feature set. For forward fill, the experimental setup can also be an influence for success, but the second major influence is that the introduction of previous values can be very deceiving to models. So, this should not occur too often, or even better, such values should be recognised.

We measure performance with and without NaN-filling, using different parameter setups and feature sets, where each setup is done using both the first and the last ten seconds of a ballwin event (as explained in Section 4.1.5). We use setups with one, five and ten estimator LSTM boosting, and use the Base and Full feature sets. We also check the performance by the baseline methods on these parameter settings, to validate that performance patterns are not limited to LSTMs. This experiment shows the performance gain by introducing new (NaN-rich) features, with a larger but less reliable dataset.

5.3.1 Results - LSTMs

Figure 6 and Figure 7 present performance of different parameter setups, focusing especially on using one, five or ten estimator LSTM boosting for NaN-filling compared to using no NaN-

filling.

In these experiment setups, average precision for feature set Base, for any number of estimators, is similar with and without NaN-filling. The models with 5 and 10 estimators without NaN-filling perform better than the same setup with NaN-filling. When no boosting is used, using NaN-filling seems to outperform not using NaN-filling. When considering the patterns of using 1, 5 or 10 estimators with and without NaN-filling, using the last 10 seconds of ballwin events diverges more when not using NaN-filling.

NaN-filling performs worse than not using NaN-filling in most setups. NaN-filling on the Full featureset results in the worst average precision. When compared, Forward Fill outperforms Constant Value Fill in most situations. For feature set Full, results are inconsistent, especially when using the first ten seconds of ballwin events. Still, model outperforms a model that simply guesses based on the ratio of positives in the dataset.

5.3.2 Results - Baseline Methods

For the baseline methods, results can be found in Figure 8 for using the first 10 seconds of a ballwin event, and in Figure 9 for using the last 10 seconds. The results of using NaN-filling are visually almost equal to not using NaN-filling for feature set Base, for most baseline methods.

Using baseline methods using feature set Full, average precision is increasing over time when using NaN-filling, and is relatively constant. With NaN-filling, feature set Full generally outperforms or gets similar results to similar feature set Base setup. Also, it correlates more to setups with feature set Base than to feature set Full without NaN-filling. Forward Fill and Constant Value NaN-filling achieve almost identical results in most setups.

Overall, Naive Bayes is one of the least reliable and worst performing baseline methods in every setup, with and without NaN-filling. AdaBoost Classifier, Random Forrest Classifier and Logistic Regression all perform best in a subset of the experiment setups. The best performance is still achieved by Logistic Regression using the Full feature set without NaN-filling. This feature set achieves the best results in most setups. The second best performing feature set is Full with NaN-filling.

5.3.3 Results - LSTMs vs Baseline Methods

Baseline methods still achieve similar or higher average precision in most equal setups, even when using NaN-filling. Compared to the results from Section 5.1, the best performing model does not change. NaN-filling has limited impact on feature set Base, with a slight decrease in performance. However, the worst performing models are clearly LSTMs with NaN-filling and feature set Full.

5.3.4 Discussion

The best average precision is still achieved by Logistic Regression, using feature set Full and no NaN-filling. However, the results from Section 5.2 indicated that the results of that experiment setup might not be reliable. Our second best results were achieved using feature set Full with

NaN-filling. When compared, Forward Fill achieved better results than Constant Value Fill, which is possibly due to the Forward Filled value impacting predictions more than recognising NaNs would.

We observed that most LSTMs and baselines methods did not perform better when introducing NaN-filling. For feature set Base, differences were generally very small. This probably means that either it was easier to detect NaN-filling in those models, few NaNs had to be filled, or there was enough data to compensate for these less reliable sequences. For feature set Full, impact of NaN-filling was large, with NaN-filling drastically decreasing average precision for the LSTM. For the baseline methods, average precision increased with the introduction of the new features, which suggests that we successfully applied NaN-filling.

It is also interesting to observe that when not boosting, NaN-filling performs better than no CV NaN-filling for feature set Base. This could indicate that CV NaN-filling can be successful, if its effect is not countered by boosting, or used with NaN-rich features.

5.4 Experiment IV - Sampling

In imbalanced datasets, examples from the minority class occur less frequent, which can result in models not being able to properly learn the features that identify minority class examples. Datasets in general can be too small to train models with. In this experiment, we examine undersampling and oversampling, which are explained in Section 4.1.3, as method to reduce dataset imbalance and increase dataset size, without adjusting the data. In Experiment I, Section 5.1, we saw that for certain setups, not enough data was available. Therefore, sampling could make results for those setups more reliable or even increase performance.

In this experiment, we should again consider different effectiveness for different parameter setups. However, performing the sampling methods for every model will be very time consuming, and challenging to present clearly. We are mostly interested in the effect of sampling on the dataset that was considered unreliable because of a lack of data, and its effect on our best performing dataset. Therefore, we examine the effect of using undersampling, oversampling, and both on featureset Full only. We will also limit ourselves to using the last ten seconds of ballwin events. The desired sampling strategy is set to 0.25 when using either undersampling or oversampling. When used together, we first use 0.25 as undersampling strategy, and then 0.4 as oversampling ratio.

5.4.1 Results

In Figure 12, we find the results from using different types of sampling on feature set Full. We find limited effect of using the sampling methods. Only undersampling seems to have a clear positive impact on the LSTM methods. A similar Figure for using no NaN-filling was produced. However, the sampling was not strong enough to make results more constant, which resulted in a very unclear Figure 13.

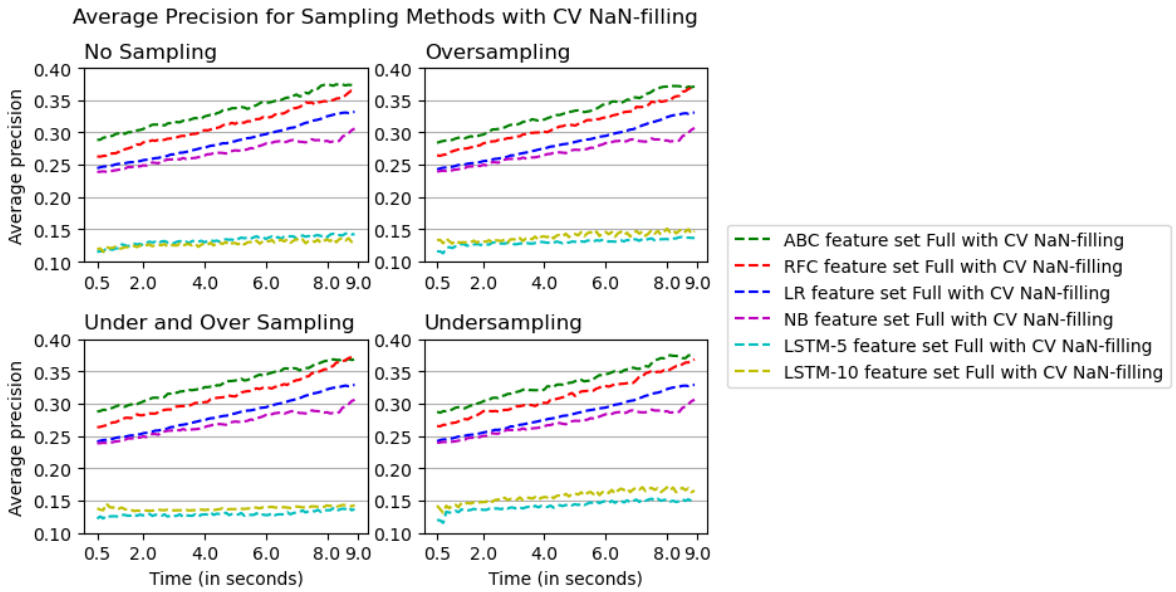


Figure 12: Average precision for feature set Full with different sampling strategies. The last 10 seconds of ballwin events from 266 matches are used, with NaN-filling.

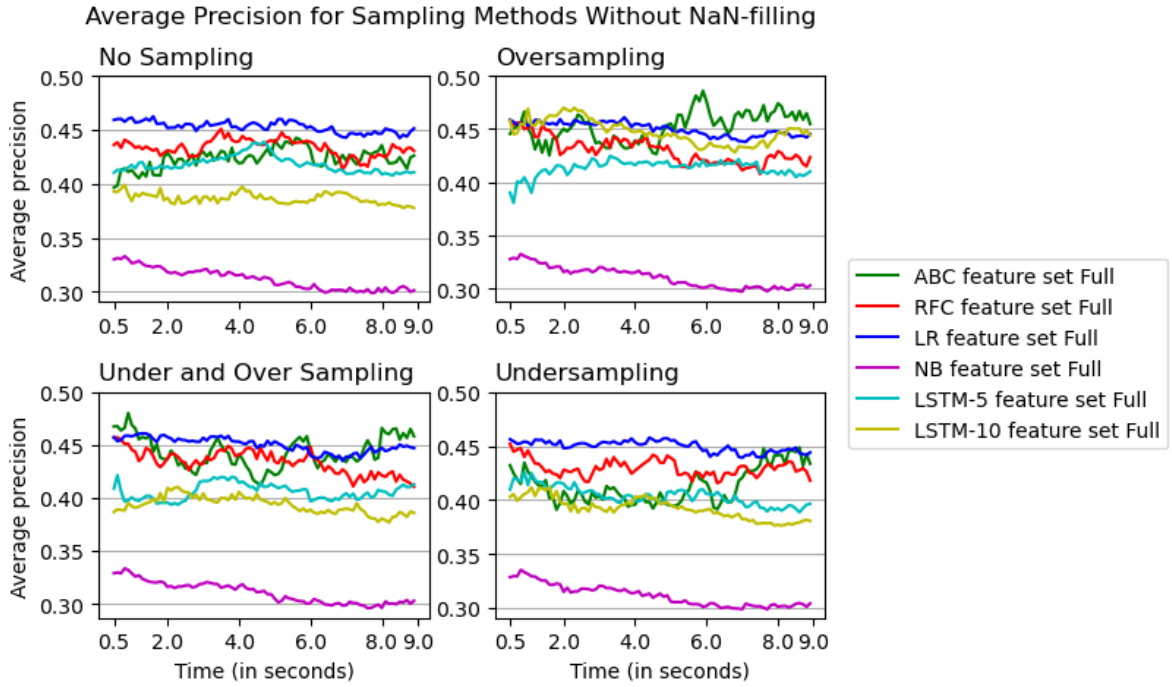


Figure 13: Average precision for feature set Full with different sampling strategies. The last 10 seconds of ballwin events from 266 matches are used, without NaN-filling.

5.4.2 Discussion

Sampling appears to have almost no effect on feature set Full. This could be the result of low sampling strategies. We could perform the same experiments with more aggressive sampling, but this would result in a dataset unrepresentative for the real world. It is also possible that performance should be at a higher level before sampling would have any effect. The positive

effect of undersampling on LSTMs could indicate that a more balanced dataset does increase performance of the model. However, it is as likely that the model is better in predicting positive examples, as it only improves performance for this specific setup. Therefore, we state that sampling did not succeed in either getting more reliable results by making the feature set larger, or increasing performance by increasing training data.

6 Conclusion

In this thesis, we tried to identify dangerous ball possessions using LSTMs, AdaBoost Classifier, Random Forest Classifier, Logistic Regression and Naive Bayes. We tried to increase performance by adding a large set of new features. These features included NaN-rich features which, though potentially informative, would generally be considered difficult to use due to their sparsity. We showed that adding these features, without additional processing and by removing sequences containing NaNs, would result in unreliable performance. This is the result of underfitting and the dataset not being representative for the problem of identifying real-world dangerous possessions. However, by using constant value NaN-filling and forward fill NaN-filling, we can use the new features reliably.

Overall, the baseline methods were generally more effective than (boosted) LSTMs, and all models performed better than guessing. In all methods that were considered reliable, we see a pattern of increasing performance over time. We also see that in this dataset, danger events generally occur at the end of a ballwin event. This correlation could imply that we can better identify dangerous ball possessions the closer we get to dangerous events. In that case, these models at least show limited comprehension of what constitutes danger.

We showed that boosting in LSTMs proved beneficial. The number of estimators that resulted in the best performance depended on the experiment setup. However, it appeared that when the problem was more difficult, a larger quantity of estimators performed better, but a smaller set of estimators could be more effective when the LSTM better understood what constituted danger in identifying dangerous LSTMs.

In future work, we could focus on the impact of subsets of feature. We now have access to a large library of features, it would be beneficial to the model to optimise which features we want to use. Our suggestion would be to start by using a feature importance method to approximate the impact of single features. In a later stage, we could use automatic feature selection methods to get better feature sets. Both methods present more reliable feature importance estimations when a model performs better, as more complicated indicators for danger would be detected. Therefore, it might be a better starting point to reconsider optimising the prediction model, as the current architecture is basic. A totally different approach would be to change the current input format. Previous work (Section 3.2) showed that by using grid-formatted input, we can visualise important spaces on the field. This input could also be using in our prediction model. In a later stage, we could reconsider our success measure and the retrieval of time frames using ballwins. Some simplification on the concepts of danger and interesting time intervals were needed to present a proof of concept. As these compromises could create biases, reconsideration of the success measure and used time intervals could improve real-world and real-time applicability.

In conclusion, in this thesis, we tried to identify dangerous soccer sequences. We presented a method to use features that were previously considered too NaN-rich. Using these features, we increased performance of previous classification models, where our baseline methods outperformed boosted LSTMs in most experiment setups. We showed that our models show at least some comprehension on what constitutes danger.

References

[1] Sarah Rudd. A framework for tactical analysis and individual offensive production assessment in soccer using markov chains. In *New England Symposium on Statistics in Sports. http://nessis. org/nessis11/rudd. pdf*, 2011.

[2] Francisco Peralta Alguacil, Pablo Piñones Arce, David Sumpter, and Javier Fernandez. Seeing in to the future: using self-propelled particle models to aid player decision-making in soccer. In *Proceedings of the MIT Sloan Sports Analytics Conference*, 2020.

[3] Floris R Goes, Matthias Kempe, Laurentius A Meerhoff, and Koen APM Lemmink. Not every pass can be an assist: a data-driven model to measure pass effectiveness in professional soccer matches. *Big data*, 7(1):57–70, 2019.

[4] Joachim Gudmundsson and Michael Horton. Spatio-temporal analysis of team sports. *ACM Computing Surveys (CSUR)*, 50(2):1–34, 2017.

[5] Javier Fernandez and Luke Bornn. Wide open spaces: A statistical technique for measuring space creation in professional soccer. In *Sloan Sports Analytics Conference*, volume 2018, 2018.

[6] D Memmert and R Rein. Match analysis, big data and tactics: current trends in elite soccer. *German Journal of Sports Medicine/Deutsche Zeitschrift fur Sportmedizin*, 69(3), 2018.

[7] Laurentius Antonius Meerhoff, Floris R Goes, A Knobbe, et al. Exploring successful team tactics in soccer tracking data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 235–246. Springer, 2019.

[8] Jan Van Haaren, Siebe Hannosset, and Jesse Davis. Strategy discovery in professional soccer match data. In *Proceedings of the KDD-16 Workshop on Large-Scale Sports Analytics*, pages 1–4, 2016.

[9] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[10] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Action classification in soccer videos with long short-term memory recurrent neural networks. In *International Conference on Artificial Neural Networks*, pages 154–159. Springer, 2010.

[11] Joris Verpalen. *Predicting player movements in soccer using Deep Learning*. PhD thesis, Tilburg University, 2019.

[12] S. Green. Assessing the performance of premier league goalscorers. `Available:http://www.optasportspro.com/about/optaproblog/posts/2012/blog-assessing-the-performance-of-premier-league-goalscorers/`, Apr 2012. [Accessed 18 Nov 2020].

[13] Tom Decroos, Lotte Bransen, Jan Van Haaren, and Jesse Davis. Actions speak louder than goals: Valuing player actions in soccer. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1851–1861, 2019.

[14] Wouter Frencken, Koen Lemmink, Nico Delleman, and Chris Visscher. Oscillations of centroid position and surface area of soccer teams in small-sided games. *European Journal of Sport Science*, 11(4):215–223, 2011.

[15] Panna Felsen and Patrick Lucey. Body shots: Analyzing shooting styles in the nba using body pose. In *Proceedings of the MIT Sloan Sports Analytics Conference, Possession Sketches: Mapping NBA Strategies, Boston, MA, USA*, pages 3–4, 2017.

[16] FR Goes, LA Meerhoff, MJO Bueno, DM Rodrigues, FA Moura, MS Brink, MT Elferink-Gemser, AJ Knobbe, SA Cunha, RS Torres, et al. Unlocking the potential of big data to support tactical performance analysis in professional soccer: A systematic review. *European Journal of Sport Science*, pages 1–16, 2020.

[17] Hugo Sarmento, Rui Marcelino, M Teresa Anguera, Jorge CampaniÇo, Nuno Matos, and José Carlos LeitÃo. Match analysis in football: a systematic review. *Journal of sports sciences*, 32(20):1831–1843, 2014.

[18] Javier Fernández, Luke Bornn, and Dan Cervone. Decomposing the immeasurable sport: A deep learning expected possession value framework for soccer. In *13th MIT Sloan Sports Analytics Conference*, 2019.

[19] Daniel Link, Steffen Lang, and Philipp Seidenschwarz. Real time quantification of dangerousity in football using spatiotemporal tracking data. *PloS one*, 11(12):e0168768, 2016.

[20] Hugo Folgado, Koen APM Lemmink, Wouter Frencken, and Jaime Sampaio. Length, width and centroid distance as measures of teams tactical performance in youth football. *European journal of sport science*, 14(sup1):S487–S492, 2014.

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

[22] W. Leeftink. Classifying sequences of football play using boosted lstms. Master's thesis, Leiden Institute of Advanced Computer Science (LIACS), The Netherlands, 2020.

[23] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, 18(1):559–563, 2017.

[24] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[25] Uwe Dick and Ulf Brefeld. Learning to rate player positioning in soccer. *Big data*, 7(1):71–82, 2019.

[26] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[27] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[28] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[29] M. Phi. Illustrated guide to lstm's and gru's: A step by step explanation. `Available:https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21`, Sept 2018. [Accessed 5 Feb 2021].

[30] C. Olah. Understanding lstm networks. `Available:http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Aug 2015. [Accessed 5 Feb 2021].

[31] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.