OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Neural Network vs. GRU in League of Legends Match Outcome Prediction: A Data-Centric Perspective

Bachelorarbeit
von

Moritz Palm
Matrikelnummer: 3281253

Fakultät Informatik und Mathematik
Ostbayerische Technische Hochschule Regensburg
(OTH Regensburg)

Gutachter: Prof. Dr. Brijnesh Jain
Zweitgutachter: Prof. Dr. Timo Baumann

Abgabedatum: 26. Dezember 2023

Herr
Moritz Palm
Konrad-Adenauer-Allee 55
93051 Regensburg

Studiengang: Künstliche Intelligenz & Data Science

1. Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum des Freistaates Bayern übergeht.

2. Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtlich und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den 26. Dezember 2023

Moritz Palm

Inhaltsverzeichnis

1 Introduction

League of Legends (LoL), developed by Riot Games, is a prominent Multiplayer Online Battle Arena (MOBA) game, a sub genre of real-time strategy games characterized by two teams of five players, known as 'summoners', competing against each other [1]. Each player controls a unique character, or 'champion', and the objective is to defeat the opposing team. LoL stands out in the MOBA genre for its global popularity, attracting millions of players and a significant viewership in professional esports tournaments [2]. While sharing core gameplay elements and map layouts with other MOBA games, LoL distinguishes itself through its diverse range of champions, abilities, and graphical styles. This thesis will therefore focus on League of Legends, given its influential status in the MOBA genre.

esports is highly relevant due to it being a huge and strongly growing market. In 2019, the esports industry's market size was valued at approximately 25B USD [3]. Esports and mobas in particular are hard to understand and follow. A live game prediction view can help fans understand the action and decisions made better and help immerse the audience by detecting upsets and swings in win probability. many games are hard to understand, due to lots of information being displayed with very little explanation a win prediction graph can help viewers understand the action and the significance of certain plays better, thus increasing engagement and enjoyment. riot games has already implemented their own proprietary win prediction a win prediction model can also help players make more informed decisions about what the optimal path of actions is

the model should be able to answer the question, if team a is far enough ahead to win or if team b with their hyper scaling heroes can come back and win

2 Related work

Utilizing machine learning methods to extract information from data generated by e-sport games is an area of ongoing research. A lot of scientific research focuses on the similar MOBA DotA 2, which has easier and more fine-grained data collection methods (see section 4.1). Due to the high similarity between these two games, it is to be expected that any findings for one game can be replicated and used for the other game with minimal adaptations. Nevertheless, to ensure a fair comparison, both games are presented separately below.

In DotA 2, a wide variety of algorithms have been used. Yu et al. [15] trained a Recurrent Neural Network (RNN) on 71,355 matches and achieved an accuracy of 0.7083 at the half-way point of a match, which according to their analysis is on average at 20 minutes. Wang [16] compared Logistic Regression with a Feedforward Neural Network (FNN) trained on up to 911,468 matches, with Logistic Regression (LR) achieving a slightly better accuracy (0.6104) than the FNN (0.588).

Silva et al. have used RNNs to predicting the winner using data of different time intervals. They achieved an accuracy of 75% when using data from between the 10 and 15 minute mark. An evaluation of LSTM resulted in lower accuracy, most likely due to the large amount of data required [6].

Author	Games	Features	Time	Accuracy
Shen [4]	10,000	5	10 min	0.726
Bahrololloomi et al. [5]	2,901	15	pre-game	0.86
Silva et al. [6]	7,621	52	up to 25 min	0.835 (25 min)
Mondal et al. [7]	296	5	post game	-
Costa et al. [8]	2,840	50	pre-game	-
Hitar-García et al. [9]	7583	26	pre-game	0.683
Zhang [10]	10,000	38	10 min	0.723
White et al. [11]	87,743	?	pre-game	0.721
Bailey [12]	671	28	15 min	0.76
Do et al. [13]	5,000	44	pre-game	0.751
Ani et al. [14]	1,500	97	pre-game	0.955

Table 1: Comparison of different works on League of Legends win prediction

3 Background

3.1 League of Legends

LoL is played with 5 players on each team on a map which is bifurcated into two bases, each linked by three lanes and housing a crucial structure called the 'nexus', which is protected by turrets. The game's primary goal is to destroy the opposing team's nexus. The map includes a jungle area in between the lanes with neutral monsters and two significant creatures, Baron Nashor and the Dragon, offering team-wide benefits when defeated. Players must accumulate gold and experience points (xp) through defeating minions, neutral monsters, or enemy champions. These in-game currencies are essential for purchasing items and levelling up, thereby augmenting a champion's capabilities.

Player roles in LoL are typically assigned with one player in the top lane, one in the mid lane, two in the bottom lane, and one in the jungle, facilitating strategic diversity and role specialization. Players select from a roster of 165 champions, each with unique abilities and characteristics, to compete in matches. Champion selection is a pivotal element of LoL gameplay, requiring players to consider team composition, damage types, assigned roles, and personal proficiency with specific champions. The theoretical number of possible champion combinations in a game is $\binom{165}{10} = 3.21 \times 10^{15}$. Although this number is quite a bit smaller in reality as not every champion can play every role and most players are only proficient with 15-20 champions [17], this underscores the game's strategic depth.

Each year, LoL introduces a new 'season', bringing substantial changes, and Riot Games issues bi-weekly patches to adjust champion balance, influencing the prevailing game strategies, or 'meta'. These patches can also include the release of a new champion or the rework of an old one. Frequent changes force players to be able to quickly adapt and learn new champions and mechanics.

To evaluate player skill, LoL utilizes a proprietary rating system, probably a modified Elo system [18]. This system ensures that players are matched with and against others of comparable skill levels, maintaining competitive balance and fairness in the game.

3.2 Neural Networks

Artificial Neural Networks (ANNs) are computational models that emulate the processing patterns of the human brain. The fundamental computational unit of an ANN is the neuron, a concept first proposed by McCulloch et al. [19].

A neuron computes an output activation a from a set of input values $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, where m denotes the number of inputs. The neuron's weighted input z is calculated as the dot product of the input vector $\mathbf{x}$ and the weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_m)$, plus

a bias term b:

$$z = \sum_{i=1}^{m} w_i x_i + b = \mathbf{w}^\top \mathbf{x} + b. \tag{1}$$

The weighted sum z is then passed through an activation function ϕ, such as a sigmoid or Rectified Linear Unit (ReLU), to introduce non-linearity:

$$a = \phi(z) = \phi(\mathbf{w}^\top \mathbf{x} + b). \tag{2}$$

The Multi-Layer Perceptron (MLP), introduced by Rosenblatt [20], organizes neurons into layers. Data flows from the input layer, through one or more hidden layers, to the output layer. In a fully connected feed-forward network, the computation in each layer l is:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \tag{3}$$

where $\mathbf{a}^{(l)}$ represents the activation of layer l, $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ the bias vector. The vector $\mathbf{z}^{(l)}$ is then passed through the activation function for layer l, which is applied elementwise:

$$\mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)}). \tag{4}$$

The output layer L produces the network's prediction $\hat{\mathbf{y}}$. The choice of activation function is dependent on the task, a common choice for classification is the softmax function [21]

$$S(\mathbf{x}_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{5}$$

where K is the number of classes and $i = 1, ..., K$. The softmax function returns a probability distribution over the predicted output classes.

To approximate any measurable function, an ANN requires at least one hidden layer [22]. The network's weights and biases are adjusted during training to minimize a loss function E. Common loss functions include Mean Squared Error (MSE) for regression tasks:

$$E_N = \frac{1}{N} \sum_{k=1}^{N} (y_k - \hat{y}_k)^2, \tag{6}$$

and Cross-Entropy Loss (CEL) for multi-class classification tasks:

$$E_N = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log(\hat{y}_{nk}) \tag{7}$$

where N is the number of samples, y_{nk} is the (one-hot encoded) ground truth and $\hat{y}_{nk}$ is the softmax output.

Backpropagation [23] is a key algorithm for training ANNs, involving a forward pass to compute activations and a backward pass to compute gradients. The gradients of the loss function with respect to the weights and biases are computed using the chain rule of calculus. For a given layer l, the gradient of the loss E with respect to the weights $\mathbf{W}^{(l)}$is

$$\Delta \frac{\partial E}{\partial \mathbf{W}^{(l)}} = \frac{\partial E}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}}, \tag{8}$$

and with respect to the bias $\mathbf{b}^{(l)}$

$$\frac{\partial E}{\partial \mathbf{b}^{(l)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \tag{9}$$

as

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = 1 \tag{10}$$

where $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$and $\mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)})$. The gradients are then used to update the weights and biases, typically using an optimization algorithm like gradient descent. The weights are updated via:

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta \frac{\partial E}{\partial \mathbf{W}^{(l)}} \tag{11}$$

where η is the learning rate.

Through iterative forward and backward propagation, the network gradually converges to a state where the loss is minimized, indicating successful learning of the patterns in the data.

3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) extend the capabilities of feed-forward neural networks to handle sequential data by introducing the concept of recurrence. In an RNN, the output at each time step is influenced not only by the current input but also by the network's previous internal state, known as the hidden state. This design enables RNNs to capture temporal dependencies, making them particularly effective for tasks involving sequential data, such as speech recognition and natural language processing [24]. The concept of a fully connected RNN was first proposed by Elman [25].

RNNs maintain a 'state vector' in their hidden units, which implicitly contains information extracted from all past elements of the sequence [24]. The hidden state $\mathbf{h}_t$ at time step t is updated as follows:

$$\mathbf{h}_t = \begin{cases} 0, & \text{if } t = 0, \\ \sigma_h(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}_h) & \text{otherwise,} \end{cases} \tag{12}$$

where $\mathbf{U}$ is the weight matrix for the hidden state and $\mathbf{W}$ is the weight matrix for the input. A common choice for σ_h is the tanh function. In a single-layer RNN, all weight matrices $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ (and biases) are shared across timesteps. The output $\hat{\mathbf{y}}_t$ of an RNN at time step t can be calculated as:

$$\begin{aligned} \mathbf{o}_t &= \mathbf{V} \cdot \mathbf{h}_t + \mathbf{b}_\mathbf{h} \\ \hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{o}_t) \end{aligned} \tag{13}$$

where $\mathbf{V}$ is the weight matrix associated with the cell output.

The loss over T timesteps is defined by

$$E_T = \frac{1}{T} \sum_{t=1}^{T} l(\hat{\mathbf{y}}_t, \mathbf{y}) \tag{14}$$

where $l(\hat{\mathbf{y}}_t, \mathbf{y})$ is the loss at timestep t.

Backpropagation Through Time (BPTT) unfolds the RNN across time steps (see Figure 1) and applies the backpropagation algorithm. In order to train U, V and W, we need their respective gradients $\frac{\delta E}{\delta \mathbf{U}}$, $\frac{\delta E}{\delta \mathbf{V}}$ and $\frac{\delta E}{\delta \mathbf{W}}$.

As the weight matrices are shared across timesteps, we can generally sum the gradients from each timestep t together. The gradient of the loss function with regards to the output matrix V does not depend on the hidden state h_t and can thus be calculated easily.

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{V}} &= \sum_{t}^{T} \frac{\partial E_t}{\partial \mathbf{V}} \\ &= \sum_{t}^{T} \frac{\partial E_t}{\partial \hat{\mathbf{y}}_t} \cdot \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{V}}\end{aligned} \tag{15}$$

Now we consider the gradient with respect to the weight matrix for the hidden state U at the time step $t+1$:

$$\frac{\partial E_{t+1}}{\partial \mathbf{U}} = \frac{\partial E_{t+1}}{\partial \hat{\mathbf{y}}_{t+1}} \frac{\partial \hat{\mathbf{y}}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{U}} \tag{16}$$

As the hidden state h_{t+1} depends on the hidden state of the previous timestep h_t, we need to recursively calculate the partial derivatives of all the previous timesteps, yielding the following formula:

$$\frac{\partial E_{t+1}}{\partial \mathbf{U}} = \sum_{k=1}^{t+1} \frac{\partial E_{t+1}}{\partial \hat{\mathbf{y}}_{t+1}} \frac{\partial \hat{\mathbf{y}}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{U}} \tag{17}$$

Applying the chain rule to $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k}$ yields

$$\frac{\partial E_{t+1}}{\partial \mathbf{U}} = \sum_{k=1}^{t+1} \frac{\partial E_{t+1}}{\partial \hat{\mathbf{y}}_{t+1}} \frac{\partial \hat{\mathbf{y}}_{t+1}}{\partial \mathbf{h}_{t+1}} \left(\prod_{j=k}^{t} \frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{h}_j} \right) \frac{\partial \mathbf{h}_k}{\partial \mathbf{U}} \tag{18}$$

[26]. Summing the partial derivatives over timesteps similar to equation (15) yields the full equation

$$\frac{\partial E}{\partial \mathbf{U}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial E_{t+1}}{\partial \hat{\mathbf{y}}_{t+1}} \frac{\partial \hat{\mathbf{y}}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{U}} \tag{19}$$

where

$$\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} = \left(\prod_{j=k}^{t} \frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{h}_j} \right) = \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \ldots \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \tag{20}$$

The gradient with respect to $\mathbf{W}$ follows similarly. As first demonstrated by Bengio et al. [27], RNNs face challenges with exploding or vanishing gradients, particularly in long sequences. This can be shown by examining a single term from equation (20) as this is the partial derivative between two vectors and as such a Jacobian matrix :

$$\frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{h}_j} = \mathbf{U}^\top \mathrm{diag}(\sigma_h'(\mathbf{W}\mathbf{x}_{j+1} + \mathbf{U}\mathbf{h}_j + \mathbf{b}_h)) \tag{21}$$

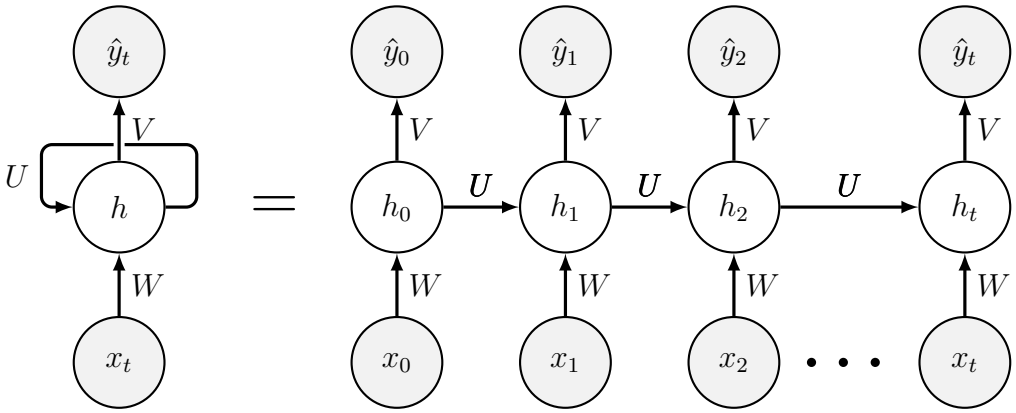

Figure 1: Unrolling of an RNN over time

where diag() converts a vector into a diagonal matrix and σ' computes the element-wise derivative of σ [28]. The eigendecomposition of $\frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{h}_j}$ yields the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ where $|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$ with their corresponding eigenvectors $v_1, v_2, ..., v_n$. The change in hidden state Δh_{j+1} in direction of a vector v_i is multiplied with the eigenvalue of this eigenvector: $\lambda_i \Delta h_{j+1}$. As these factors are multiplied across timesteps, the change is scaled by a factor equivalent to λ_i^t which scales exponentially with the timestep t. If $\lambda_1 < 1$ the gradient will vanish while if $\lambda_1 > 1$ the gradient will explode when considering $t \to \infty$ [28]. This issue hinders their ability to learn long-range dependencies [29].

3.4 GRU

In order to overcome the exploding/vanishing gradient problem of vanilla RNNs, gated networks like the Long Short-Term Memory (LSTM) [30] and Gated Recurrent Unit (GRU) [31] have been developed [32]. As they introduce an increased number of parameters compared to traditional RNNs, gated networks like the LSTM and GRU demand greater computational power [33]. Compared to the LSTM network, GRU reduces the number of gate networks to two, thus being simpler to implement and compute [31]. Chung et al. even found that GRU is at least comparable to LSTM most of the time [34]. The gates control the activation of each hidden unit. The reset gate is calculated by

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{22}$$

and the update gate z_j by

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{23}$$

[33]. The hidden state update is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h}_t$, where the update gate z_t influences how much the hidden state is changed [34]:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{24}$$

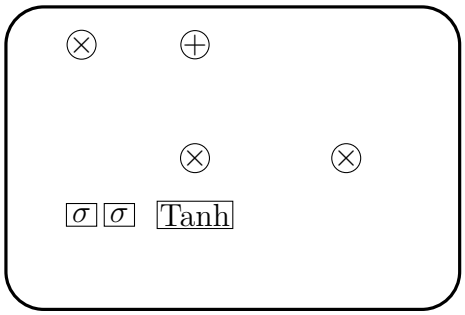

Figure 2: Gated Recurrent Unit

with

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1} + b_h) \tag{25}$$

. In equations (24) and (25) $\odot$ denotes the element-wise (Hadamard) multiplication.

3.5 Feature Selection

Feature Selection is pivotal in machine learning, particularly when dealing with high-dimensional data. It serves the primary objectives of improving model performance by mitigating the 'curse of dimensionality,' enhancing predictive accuracy, and reducing overfitting. By eliminating irrelevant or redundant features, the model's generalization capacity is enhanced, contributing to model interpretability and potentially reducing training times. Feature selection methods can be broadly categorized into three distinct types [35]:

Filter Methods These methods rely on model-invariant information, such as feature-class label correlation. They are computationally efficient but may not capture complex relationships within the data.

Wrapper Methods Wrapper methods train models iteratively on various feature subsets, incurring a higher computational cost but enabling the detection of interactions among variables.

Embedded Methods These methods perform inherent feature selection, often as an integral part of the modeling process. Tree-based models, such as Decision Trees and Gradient Boosted Trees, typically employ feature selection based on metrics like the Gini index or entropy.

Below, two different feature selection methods are discussed in detail.

3.5.1 Pearson's Correlation Coefficient

Pearson's correlation coefficient (PCC) is a statistical measure widely used to evaluate the linear relationship between two variables. Specifically, we consider its application in the context of feature selection in machine learning, where it is used to assess the linear correlation between input features and the target variable. We regard the input vector

$\mathbf{x}$ as a manifestation of an underlying, unknown distribution. Here, X_i represents the random variable corresponding to the i^{th} component of $\mathbf{x}$, and y is the target value, viewed as a realization of the random variable Y [36]. PCC is employed to quantify the linear correlation between these two random variables. It is defined by the formula:

$$R(i) = \frac{\text{cov}(X_i, Y)}{\sqrt{\text{var}(X_i) \cdot \text{var}(Y)}}, \tag{26}$$

where $\text{cov}(X_i, Y)$ is the covariance between X_i and Y, and $\text{var}(X_i)$ and $\text{var}(Y)$ are the variances of X_i and Y, respectively [37]. In order to ascertain the significance of the test results, a hypothesis test needs to be performed, with the null hypothesis being that there is no correlation between the feature and the target. While simple and effective for identifying linear relationships, PCC only captures linear dependencies and might miss non-linear relationships crucial for neural networks.

3.5.2 Gradient Boosted Trees

Gradient Boosted Trees (GBT) is an ensemble learning technique that can be used for feature selection. It builds the model in a stage-wise fashion, with each tree being added to correct the errors made by the previous ones. There are different types of importance, such as the average or total gain across all splits the feature is used in. The simplest definition is the 'weight', defined as the number of times a feature is used to split the data across all trees [38].

4 Data

As two very different experiments are compared against each other, two different datasets need to be constructed: one dataset containing all relevant information prior to the start of the game and one dataset containing only the temporal information from the beginning of the game.

4.1 Data Collection

High-Rank Matches The focus of data acquisition was directed towards high-rank matches, in which a mix of excellent amateur and professional players play. Lower rank matches are not considered due to their higher unpredictability as less skilled players make huge, game-changing mistakes way more often. Pro matches, defined as professional players playing with their respective teams in an esport tournament or league, were not included as there are way less matches and they are not available through the official Riot Games API. High rank matches in this context are defined as having at least one player holding the rank of Master, Grandmaster or Challenger. Riot Games themselves considers any rank above Diamond 3 as 'Elite' [39], but we raise this bar just slightly to only include any rank above Diamond 1. Due to the fact that for a match to be included in the dataset, only one out of ten players needs to hold one of the aforementioned highest ranks, some slightly lower ranked players are also present in the dataset. These ranks combined account for the top 0.2% of all players [40].

Riot Games API The primary source of data stemmed from the Riot Games API [41], a comprehensive repository of information pertaining to League of Legends gameplay. The Riot Games API provided access to a plethora of essential data points, including champion statistics, general match information, timeline details, and player-specific information. These variables collectively form a comprehensive and multifaceted dataset crucial for the development of an effective predictive model.

Other Data Sources However, not all pertinent data were available directly from the Riot Games API. These include the general winning chance of each champion and statistics on how each player performs on each relevant champion. To address this limitation, a web-scraping approach was employed to gather additional relevant information.

Regions Multiple regions were included in the data collection process, including Europe West (EUW), Europe Nordic & East (EUN), Korea (KR), and North America

(NA). This regional diversity contributes to the model's generalizability across different player bases and playing styles.

Period of Time All matches included in the dataset were played in season 13 and on patch 20. It is important that all matches are played on the same patch, as a patch may cause major shifts in the balance of the game, thus making certain strategies and champions way better than others.

In summary, the data collection process for this study involved a dual-pronged approach, leveraging the extensive resources provided by the Riot Games API alongside a targeted web-scraping strategy. The resulting raw dataset containing $38,573$ and $3,972$ matches in the pre-game and in-game datasets respectively, stored in a PostgreSQL Database, reflects a comprehensive compilation of high-rank amateur League of Legends matches.

4.2 Dataset Properties

The dataset under consideration encompasses a total of 38,573 matches drawn from various regions. A visual representation of the distribution of matches across these regions is provided in Figure 3. It is important to note that due to a lack of official data pertaining to the number of games played or the number of players in each region, we are unable to conclusively verify whether the distribution of matches within our dataset aligns with the true underlying distribution of games played per region.

The dataset primarily comprises matches from three major regions: North America, Western Europe, and South Korea, which collectively constitute the vast majority of matches in our dataset. Consequently, it is reasonable to assume that this composition approximately mirrors the real-world distribution of matches, with the exception of China, whose matches are not available. This assumption is further reinforced by the distribution of spots in the world championship, where these four regions are the only ones to get guaranteed spots in the main event. It is worth emphasizing, however, that while regional disparities may exist in terms of match characteristics, they are not expected to be substantial. Thus, the effect of drawing a uniform sample from the underlying distribution on the quality of our data is anticipated to be negligible.

As only matches with a game length of at least 16 minutes are collected, the shortest match is 16 minutes long, while the longest game is 59.62 minutes long. The average match length is 27.50 minutes. Figure 4 graphically illustrates the distribution of game durations. Notably, the histogram reveals a prominent spike at the 16-minute mark. This spike corresponds to the earliest possible conclusion time for a match, as League of Legends prohibits surrendering prior to the 15th minute of gameplay. In instances where an entire team collectively acknowledges the futility of their chances of victory, a surrender may be initiated at the 15-minute threshold. If a simple majority of team members want to surrender, they have to wait until the 20th minute. However, should a simple majority of team members decide to surrender, they must adhere to a 20-minute waiting period before being able to do so. Consequently, this unique feature of the game's mechanics clarifies the relatively diminished frequency of matches ending in the

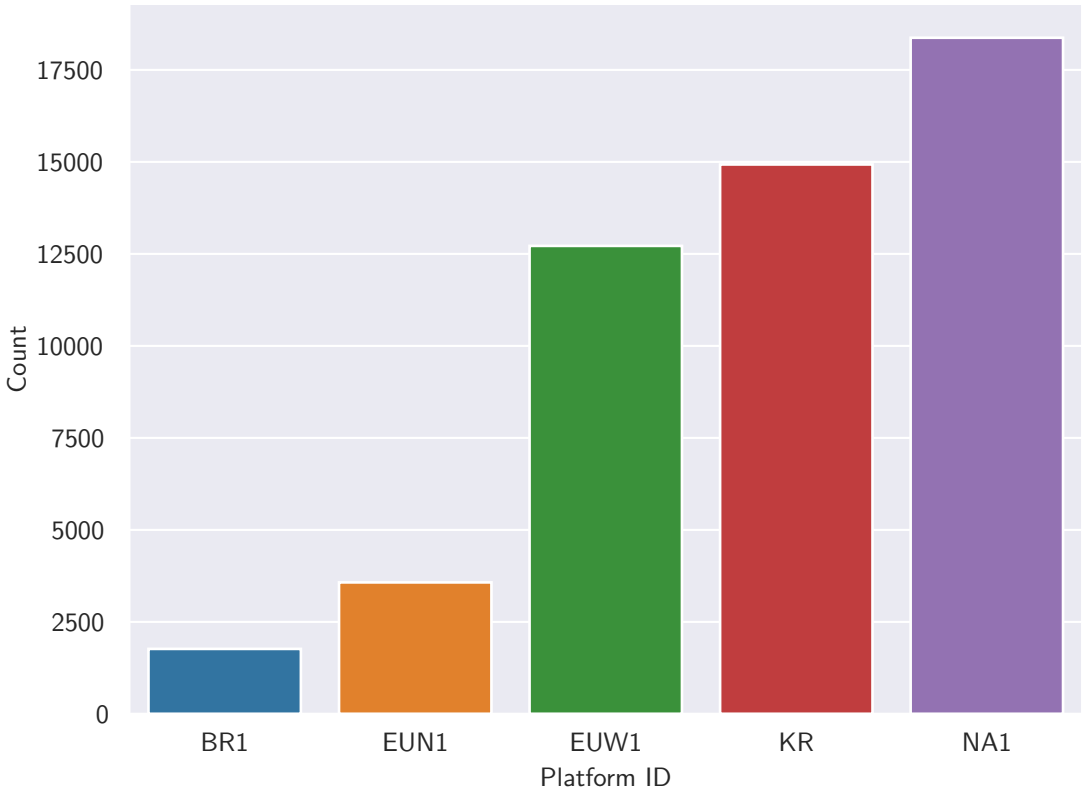

Figure 3: Region distribution of all matches in the dataset. Even though North America is most likely not the largest region, it has the largest number of games in the dataset.

17th to 19th-minute range within our dataset.

4.2.1 Pre-Game Dataset

The raw pre-game dataset contains 368 columns which can be categorized into four distinct groups: General Match Information, Player Information, Champion Information and Player-Champion Information. General match details, such as the patch number, are exclusively utilized for validation purposes and are excluded from the final dataset.

Player Information Player Information feature x_p is a two-dimensional vector containing information about the player. This includes the account level, serving as an indicator of the player's accumulated gaming experience, and the player's rank, functioning as a metric for assessing the player's skill level. transform rank info into a single float number win rate calculation

Champion Information The Champion Information feature $\mathbf{x}_c$ is composed of

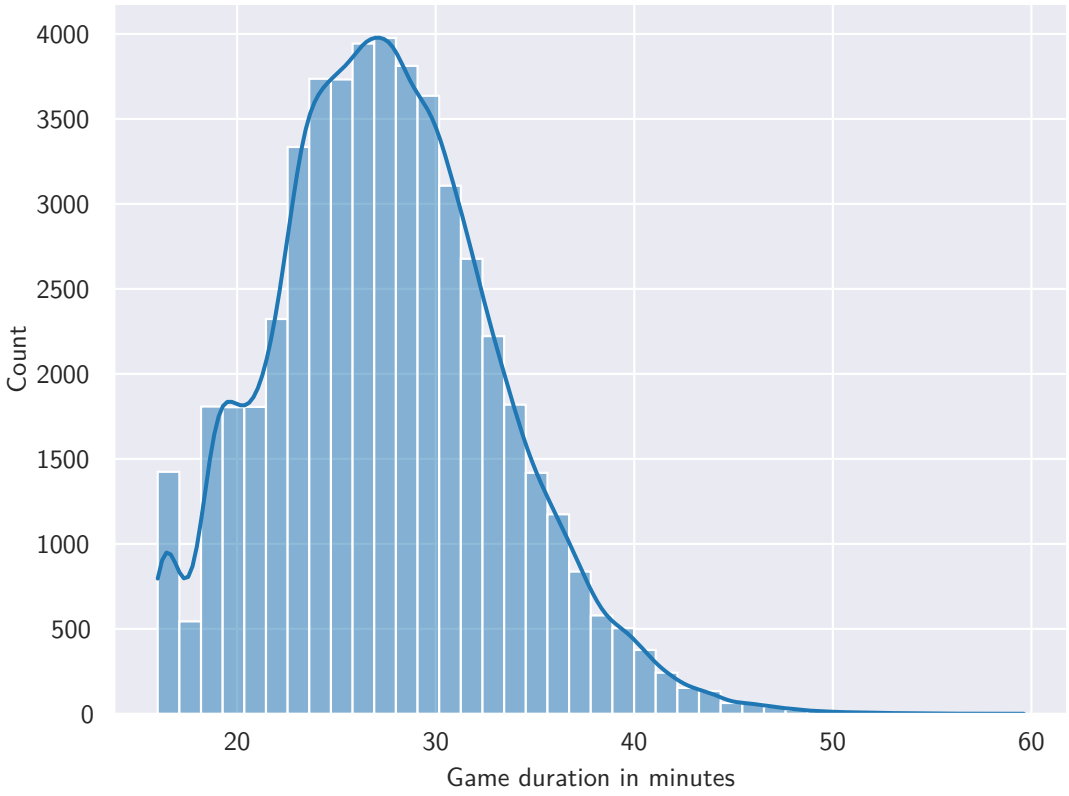

Figure 4: Distribution of game duration with its kernel density estimation. The spike at 16 minutes is explained by the fact that this is the earliest possible surrender time.

However, it is noteworthy that a limitation inherent in these metrics lies in their aggregation across all player ranks, reducing their specificity to the ranks under analysis. conversion of champion tier into integer

Player-Champion Information This feature vector contains information about the player on a specific champion. It encompasses metrics such as the average amount of gold earned by the player across all matches played on the champion during Season 13. Costa et al. [8] found that the most pivotal feature within this category is the player's win rate while piloting this champion. Unfortunately, this information is not readily available to be extracted by our web scraper, necessitating its omission from the dataset. Each feature category like 'average gold per match' is split into 10 features, one for each participant. This is a very fine-grained approach, greatly increasing the dimensionality of the dataset. For some features, a broader approach is sufficient, where statistics are averaged for each team, reducing 10 features per category to two. champion number one hot encoded

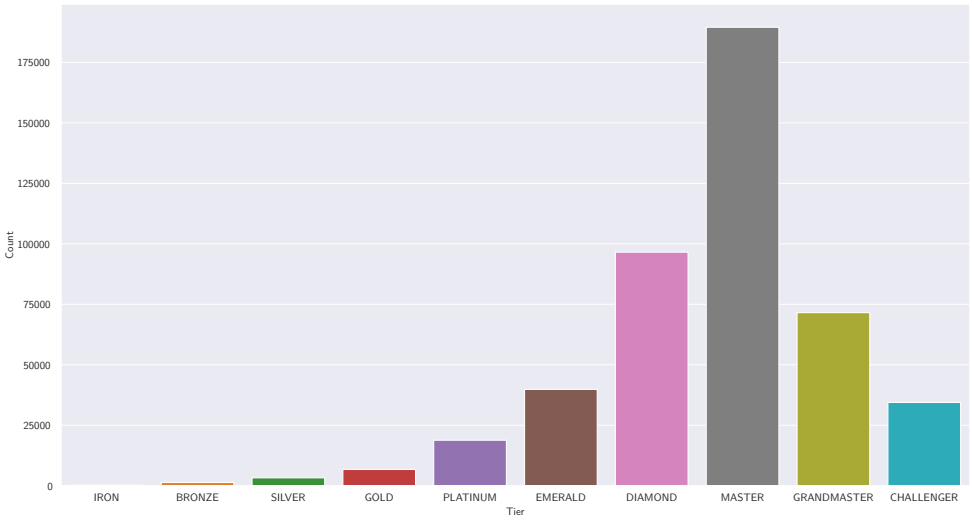

Figure 5: Distribution of ranks in the dataset

4.2.2 In-Game Dataset

The raw in-game dataset contains 381 features describing the current state of the game after every minute. It primarily includes player-specific metrics such as damage dealt to opponents, champion level, and cumulative gold. These statistics, recorded per minute, create a discrete time series. Furthermore, key events like the number of turrets destroyed and each team's total gold are tracked to more precisely gauge the game's state. Gold, a critical indicator, underscores each victory milestone - be it destroying a turret or defeating an adversary. Often, the winning team can be predicted by analysing gold trends. As illustrated in Figure 6, members of the victorious team typically amass significantly more gold by the game's conclusion compared to their counterparts.

Destroying turrets is crucial in the game, offering significant gold rewards and map control. With a minimum of five turrets required for victory, their destruction serves as a key indicator of a team's likelihood to win.

4.3 Data Processing

Scaling and Partitioning dataset is partitioned into train, validation and test set with a validation and test size of 4,000 samples corresponding to 10 percent respectively. the validation set is used to optimize hyperparameters while the test set is used to evaluate the final performance of the model in order to

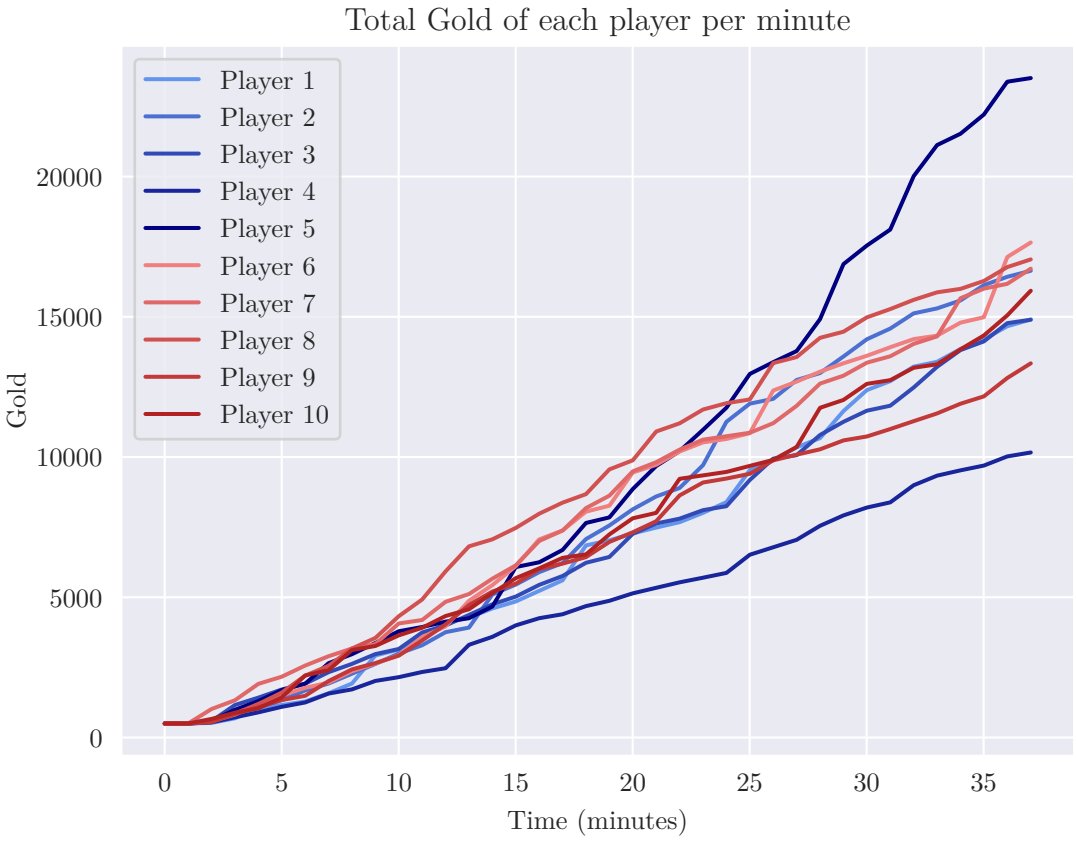

Figure 6: Total gold accumulated by each player of the course of the match, separated into teams by color. A clear separation between the blue and red team is noticeable especially at the later stages of the game.

5 Results and Discussion

5.1 Feature Selection Results

Manual feature selection using domain knowledge was performed.

As expected, the significance of the features does not differ much within a category, so the correlation coefficients are averaged per category. This enables a clearer result on the importance of feature categories.

The results of the test are found in table 2. All p-values in the table are below the significance threshold of 0.05, confirming the statistical significance of the correlation. The highest correlation is 0.15, belonging to the KDA feature. This feature measures the average ratio of $\frac{\text{kills+assists}}{\text{deaths}}$ the player achieved on his champion. This is a very low correlation, indicating that no single feature has a strong linear relationship with the target. This illustrates the need for multivariate analysis.

5.2 Pre-Game Classification

using neural network

5.2.1 Hyperparameter Optimization

5.3 Mid-Game Classification

retrain probably necessary for every patch

Feature	Correlation coefficient	GBT Feature Importance
KDA	**0.133**	**726**
Deaths	0.092	284
Assists	0.081	403
Gold	0.063	393
Kills	0.060	310
LP	0.050	541
Champion Level	0.034	50
CS Per Minute	0.027	394
Last Play Time	0.026	349
Win Rate	0.022	301
Champion Tier	0.020	66
Champion Points	0.015	401

Table 2: Average Pearson's correlation coefficient and average Gradient Boosted Tree Feature Importance for the 12 features categories with the highest absolute average correlation. Notably, the KDA feature is not only by far the most important, but its components Kills, Deaths and Assists are in the top 5 correlation coefficients as well.

6 Conclusion

Bibliography

[1] M. Mora-Cantallops and M.-Á. Sicilia. MOBA Games: A Literature Review. *Entertainment Computing* 26, 128–138, May 2018.

[2] C. Gough. League of Legends Championships Viewers 2022. URL: `https://www.statista.com/statistics/518126/league-of-legends-championship-viewers/` (visited on 12/09/2023).

[3] J. Ahn, W. Collis, and S. Jenny. The One Billion Dollar Myth: Methods for Sizing the Massively Undervalued Esports Revenue Landscape. *International Journal of Esports* 1(1), Oct. 4, 2020.

[4] Q. Shen. A Machine Learning Approach to Predict the Result of League of Legends. In: *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*, 38–45, Feb. 2022.

[5] F. Bahrololloomi, F. Klonowski, S. Sauer, R. Horst, and R. Dörner. E-Sports Player Performance Metrics for Predicting the Outcome of League of Legends Matches Considering Player Roles. *SN Computer Science* 4(3), 238, Mar. 2, 2023.

[6] A. L. C. Silva, G. L. Pappa, and L. Chaimowicz. Continuous Outcome Prediction of League of Legends Competitive Matches Using Recurrent Neural Networks. *SBC-proceedings of SBCGames*, 2179–2259, 2018.

[7] J. J. Mondal, A. Zahin, M. A. Manab, and M. Zahidul Hasan. Does A Support Role Player Really Create Difference towards Triumph? Analyzing Individual Performances of Specific Role Players to Predict Victory in League of Legends. In: *2022 25th International Conference on Computer and Information Technology (ICCIT)*, 768–773, Dec. 2022.

[8] L. M. Costa, R. G. Mantovani, F. C. Monteiro Souza, and G. Xexéo. Feature Analysis to League of Legends Victory Prediction on the Picks and Bans Phase. In: *2021 IEEE Conference on Games (CoG)*, 01–05, Aug. 2021.

[9] J. A. Hitar-García, L. Morán-Fernández, and V. Bolón-Canedo. Machine Learning Methods for Predicting League of Legends Game Outcome. *IEEE Transactions on Games* 15(2), 171–181, June 2023.

[10] Y. Zhang. Prediction of Esports Game Results Using Early Game Datasets. In: *ICMLCA 2021; 2nd International Conference on Machine Learning and Computer Application*, 1–6, Dec. 2021.

[11] A. White and D. M. Romano. Scalable Psychological Momentum Forecasting in Esports, 2020.

[12] K. Bailey. Statistical Learning for Esports Match Prediction.

[13] T. D. Do, S. I. Wang, D. S. Yu, M. G. McMillian, and R. P. McMahan. Using Machine Learning to Predict Game Outcomes Based on Player-Champion Experience in League of Legends. In: *Proceedings of the 16th International Conference on the Foundations of Digital Games*, 1–5, Oct. 21, 2021.

[14] R. Ani, V. Harikumar, A. K. Devan, and O. Deepa. Victory Prediction in League of Legends Using Feature Selection and Ensemble Methods. In: *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, 74–77, May 2019.

[15] L. Yu, D. Zhang, X. Chen, and X. Xie. *MOBA-Slice: A Time Slice Based Evaluation Framework of Relative Advantage between Teams in MOBA Games*, July 22, 2018.

[16] W. Wang. "Predicting Multiplayer Online Battle Arena (MOBA) Game Outcome Based on Hero Draft Data". MA thesis. Dublin, National College of Ireland, Dec. 21, 2016. 16 pp. URL: `https://norma.ncirl.ie/2523/` (visited on 10/05/2023).

[17] Autor unbekannt. 2022 Recap. URL: `https://yearin.lol` (visited on 12/19/2023).

[18] A. Jansson and E. Karlsson. *Neural Networks for Standardizing Ratings inLeague of Legends*. URL: `https://urn.kb.se/resolve?urn=urn:nbn:se:oru:diva-102668` (visited on 12/09/2023), 2022.

[19] W. S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The bulletin of mathematical biophysics* 5(4), 115–133, Dec. 1, 1943.

[20] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65(6), 386–408, 1958.

[21] J. S. Bridle. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In: F. F. Soulié and J. Hérault (Hrsg.). *Neurocomputing*, 227–236, 1990.

[22] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks* 2(5), 359–366, Jan. 1989.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-Propagating Errors. *Nature* 323(6088), 533–536, Oct. 1986.

[24] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature* 521(7553), 436–444, May 28, 2015.

[25] J. L. Elman. Finding Structure in Time. *Cognitive Science* 14(2), 179–211, 1990.

[26] M. M. Arat. Backpropagation Through Time for Recurrent Neural Network. URL: `https://mmuratarat.github.io//2019-02-07/bptt-of-rnn` (visited on 12/21/2023).

[27] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent Is Difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166, Mar. 1994.

[28] R. Pascanu, T. Mikolov, and Y. Bengio. On the Difficulty of Training Recurrent Neural Networks. In: *Proceedings of the 30th International Conference on Machine Learning*, 1310–1318, May 26, 2013.

[29] I. Sutskever. "Training Recurrent Neural Networks". PhD thesis. Toronto, ON, Canada: University of Toronto, 2013. URL: https://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf (visited on 11/01/2023).

[30] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation* 9(8), 1735–1780, Nov. 1997.

[31] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. URL: http://arxiv.org/abs/1406.1078 (visited on 11/03/2023).

[32] G. Van Houdt, C. Mosquera, and G. Nápoles. A Review on the Long Short-Term Memory Model. *Artificial Intelligence Review* 53(8), 5929–5955, Dec. 1, 2020.

[33] R. Dey and F. M. Salem. Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 1597–1600, Aug. 2017.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. URL: http://arxiv.org/abs/1412.3555 (visited on 09/26/2023).

[35] A. Jovic, K. Brkic, and N. Bogunovic. A Review of Feature Selection Methods with Applications. In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1200–1205, May 2015.

[36] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection.

[37] G. Chandrashekar and F. Sahin. A Survey on Feature Selection Methods. *Computers & Electrical Engineering* 40(1), 16–28, Jan. 2014.

[38] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794, Aug. 13, 2016.

[39] R. Games. /Dev: Balance Framework Update - League of Legends. URL: https://www.leagueoflegends.com/news/dev/dev-balance-framework-update/ (visited on 11/09/2023).

[40] R. Games. Ranked Tiers, Divisions, and Queues. URL: https://support-leagueoflegends.riotgames.com/hc/en-us/articles/4406004330643-Ranked-Tiers-Divisions-and-Queues (visited on 11/09/2023).

[41] Autor unbekannt. Riot Developer Portal. URL: https://developer.riotgames.com/ (visited on 12/19/2023).

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network

BPTT Backpropagation Through Time

CEL Cross-Entropy Loss

FNN Feedforward Neural Network

GBT Gradient Boosted Tree

GRU Gated Recurrent Unit

LoL League of Legends

LR Logistic Regression

LSTM Long Short-Term Memory

MLP Multi-Layer Perceptron

MOBA Multiplayer Online Battle Arena

MSE Mean Squared Error

NN Neural Network

NPC non-player character

PCC Pearson's correlation coefficient

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

xp experience points