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: 17. Januar 2024

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 17. Januar 2024

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

It can be divided into three distinct categories: pre-game win prediction can support players in choosing the champion that increases their chance of winning the most, in-game win prediction can guide players focus on their way to victory and help viewers to better understand the action. Post-game analysis does not benefit a specific game, but helps players, particularly professional players examine their strengths and weaknesses.

2 Related work

The application of machine learning techniques in interpreting data from e-sports games represents a dynamic field of research. Among the various games studied, DotA 2 and League of Legends have garnered the most attention.

2.1 Win Prediction in League of Legends

A critical factor in win prediction is the timing of data collection, as outlined in Table 1. It can be categorized into three distinct phases: pre-game, in-game, and post-game, each offering unique insights and benefits.

Pre-Game Win Prediction A variety of different methods were used to predict the winner prior to the game's commencement. A notable study by White et al. [4] incorporated a broad spectrum of pre-game features, including the concept of psychological momentum, and attained an accuracy of 0.721 using logistic regression. In comparison, Do et al. [5] limited their feature set to player-champion win rates and champion mastery points, further extracting statistical features such as the team's average player-champion win rate Applying an Artificial Neural Network (ANN) to this data yielded a notable accuracy of 0.751, which is significant given the relatively small dataset used, especially considering the results of the paper on feature selection by A dedicated feature selection study by Costa et al. [6]. They identified not only player-champion win rate but also the kill-to-death ratio for the chosen champion as the most critical features. Overall it can be assumed that lower rated games are harder to predict accurately, as more mistakes happen.

In-Game Win Prediction Silva et al. [7] trained a Recurrent Neural Network (RNN) on 7621 professional games utilizing data from varying intervals, ranging from the initial 0-5 minutes to 20-25 minutes. Their findings revealed an accuracy of 75.23% when using data from between the 10 and 15 minute mark and a maximum accuracy of 83.54% when using data from the 20-25th minute. Additionally, their research comparing Long Short-Term Memory (LSTM) networks against RNNs indicated superior performance of the latter, possibly attributable to the less complex nature of the problem or limited data availability. Bailey [8] have achieved an accuracy of 0.77 by applying logistic regression to 671 professional matches.

Post-Game Win Prediction Bahrololloomi et al. [9] have built a predictor using post-game data from professional matches achieving 86% accuracy, while Ani et al. [10]

trained a Random Forest model on a mixture of pre-, in- and post-game data for a maximum accuracy of 0.998.

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

Author	Games	Features	Time	Skill Group	Accuracy
Do et al. [5]	5,000	44	pre-game	[d]	0.751
Costa et al. [6]	2,840	50	pre-game	professional	[a]
White et al. [4]	87,743	[b]	pre-game	equidistributed	0.721
Hitar-García et al. [11]	7583	26	pre-game	professional	0.683
Lin [12]	588	2231	pre-game	Gold	0.567
Kim et al. [13]	93875	295	in-game[c]	[d]	0.738
Shen [14]	10,000	5	10 min	[d]	0.726
Zhang [15]	10,000	38	10 min	high-skilled	0.723
Bailey [8]	671	28	15 min	professional	0.76
Silva et al. [7]	7,621	52	25 min	professional	0.835
Mondal et al. [16]	296	5	post game	[d]	[a]
Bahrololloomi et al. [9]	2,901	15	post-game	[d]	0.86
Ani et al. [10]	1,500	97	post-game	professional	0.955
Lin [12]	3000	[b]	post-game	Gold	0.936

[a] This work did not build a predictor, thus no accuracy was obtained.
[b] The exact number of features is unclear.
[c] The exact timestamp where the last in-game data was obtained is unclear.
[d] The skill group(s) from which the games stem is unclear.

2.2 Win Prediction in DotA 2

is this advisable?

2.3 Win prediction in other sports

and/or this?

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

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 :

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

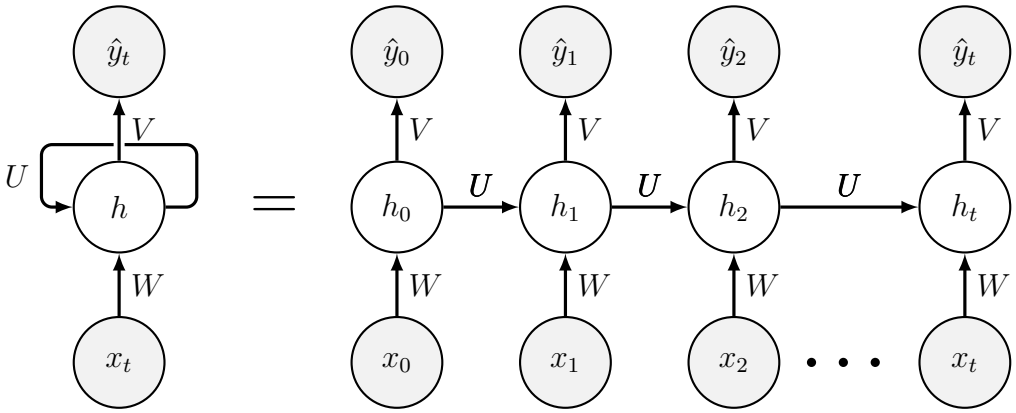

Figure 1: Unrolling of an RNN over time

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 Gated Recurrent Unit

In order to overcome the exploding/vanishing gradient problem of vanilla RNNs, gated networks like the 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], see Figure 2. 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 $\mathbf{r}_t$ is calculated by

$$\mathbf{r}_t = \sigma(\mathbf{W}_r x_t + \mathbf{U}_r h_{t-1} + \mathbf{b}_r) \tag{22}$$

and the update gate $\mathbf{z}_t$ by

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{23}$$

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

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{24}$$

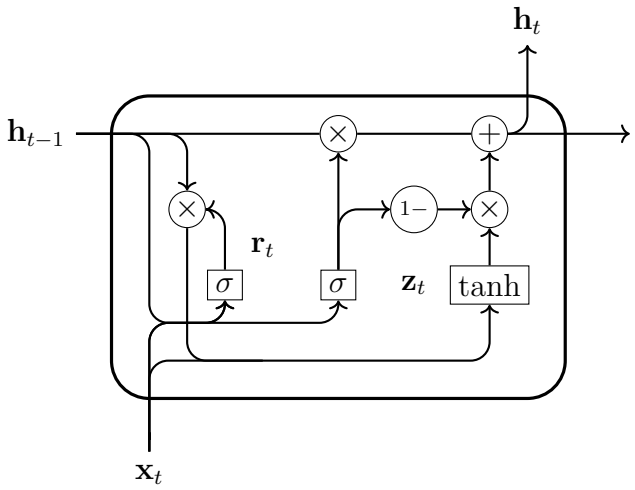

Figure 2: Gated Recurrent Unit

with

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_h). \tag{25}$$

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 and typically do not require user input in form of hyperparameters, 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.

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]. $R(i) \in [-1; +1]$ where -1 and +1 are strong negative or positive correlations and 0 indicates no correlation. 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. As it is a non-parametric model, there is no need to tune hyperparameters or risk of overfitting. While simple and effective for identifying linear relationships, PCC only captures linear dependencies and might miss non-linear relationships crucial for neural networks. Despite these limitations, PCC remains a valuable tool in feature selection for its simplicity and efficiency in revealing linear correlations.

3.5.2 Gradient Boosted Trees

Gradient Boosted Tree (GBT) is an ensemble learning technique that can be used for feature selection. The core idea of GBT is to build a model in a stage-wise fashion, where each tree incrementally improves upon the previous ones by correcting their errors. This process involves training trees sequentially, with each new tree learning to predict the residuals or errors of the previous ensemble of trees. 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

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

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.

High-Rank Matches The rating system in LoL groups players into different skill groups, where the lowest is 'Iron' and the highest 'Challenger'. The two highest ranks,'Grandmaster' and 'Challenger' contain the best 300 and 700 best players on each server. The exact number of players these tiers depend on the player number in each region (see [39]).

Similar to the methodology of Zhang, the focus of data acquisition was directed towards high-rank matches, in which a mix of excellent amateur and professional players play. High rank matches in this context are defined as having at least one player holding the rank of Master, Grandmaster or Challenger. These ranks combined account for the top 0.2% of all players [40]. Riot Games themselves considers any rank above Diamond 3 as 'Elite' [41], but we raise this bar just slightly to only include any rank at Master or above. 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.

Lower rank matches are not considered due to their higher unpredictability as less skilled players should make huge, game-changing mistakes way more often. This higher unpredictability could make it harder for the model to learn.

Pro matches, defined as professional players playing with their respective teams in an esport tournament or league, are not included as they are not available through the official Riot Games API. Professional players are still included in the dataset, but only if they played regular, non-tournament games.

Riot Games API The primary source of data stemmed from the Riot Games API [42], 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, additional relevant information was gathered by using web scraping on u.gg [43].

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.

4.2 Dataset Properties

The pre-game dataset encompasses a total of 38,573 matches, while the in-game dataset contains 28,809 matches. Below, the pre-game dataset is presented in more detail, as the smaller in-game dataset is a random sampling from the pre-game dataset.

Region Distribution The dataset is primarily comprised of matches from three major regions: North America, Western Europe, and South Korea, which collectively constitute the vast majority of matches in our dataset. 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 major regions in the dataset are the same regions getting guaranteed spots at the world championship [44] with the exception of china, whose matches are not available through the Riot Games API. Consequently, it is reasonable to assume that this composition approximately mirrors the real-world distribution of matches. A visual representation of the distribution of matches across regions is provided in Figure 3.

Game length 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

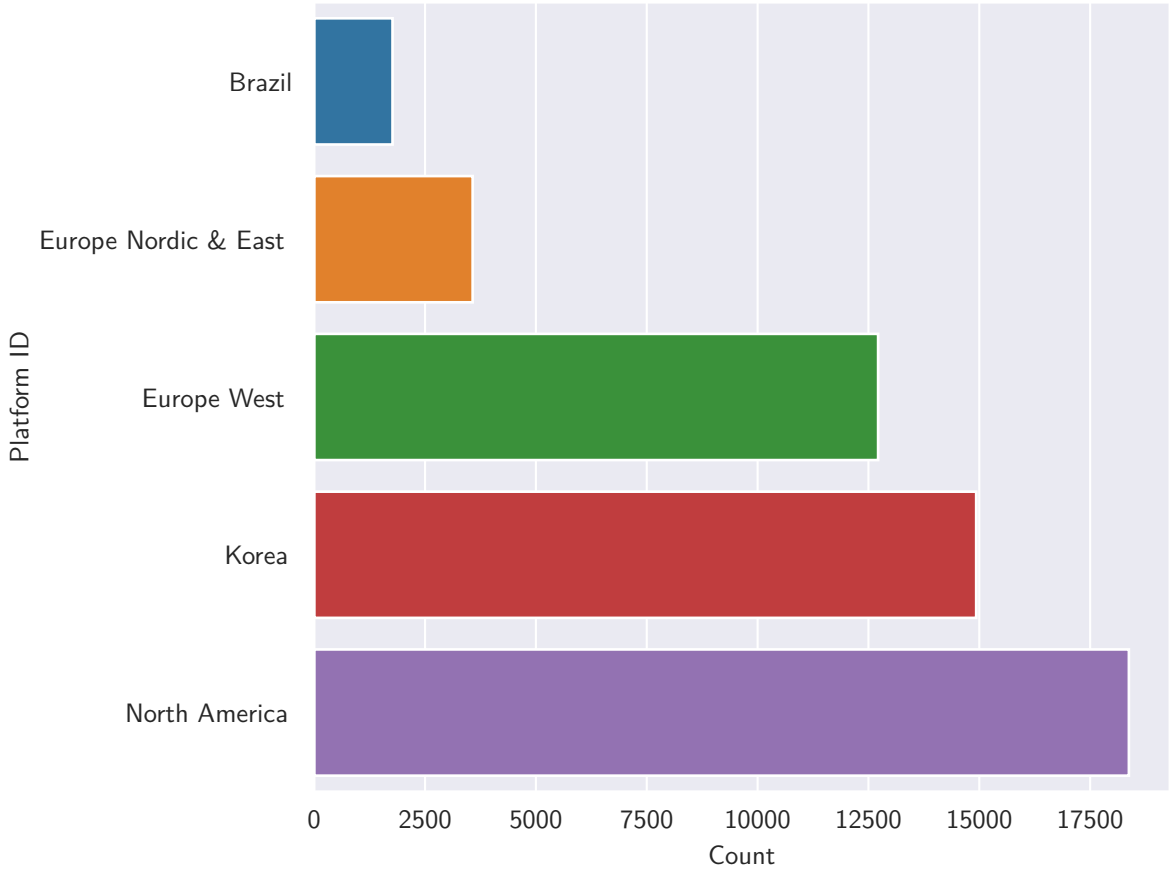

Figure 3: Region distribution

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

Rank Distribution As only games with at least one player ranked Master or above are considered, this distribution does not match the real distribution of ranks. This does introduce a bias and makes the findings less applicable to games in lower ranks. As argued in 4.1, lower rank games could make the learning harder due to higher unpredictability.

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 information, such as the patch

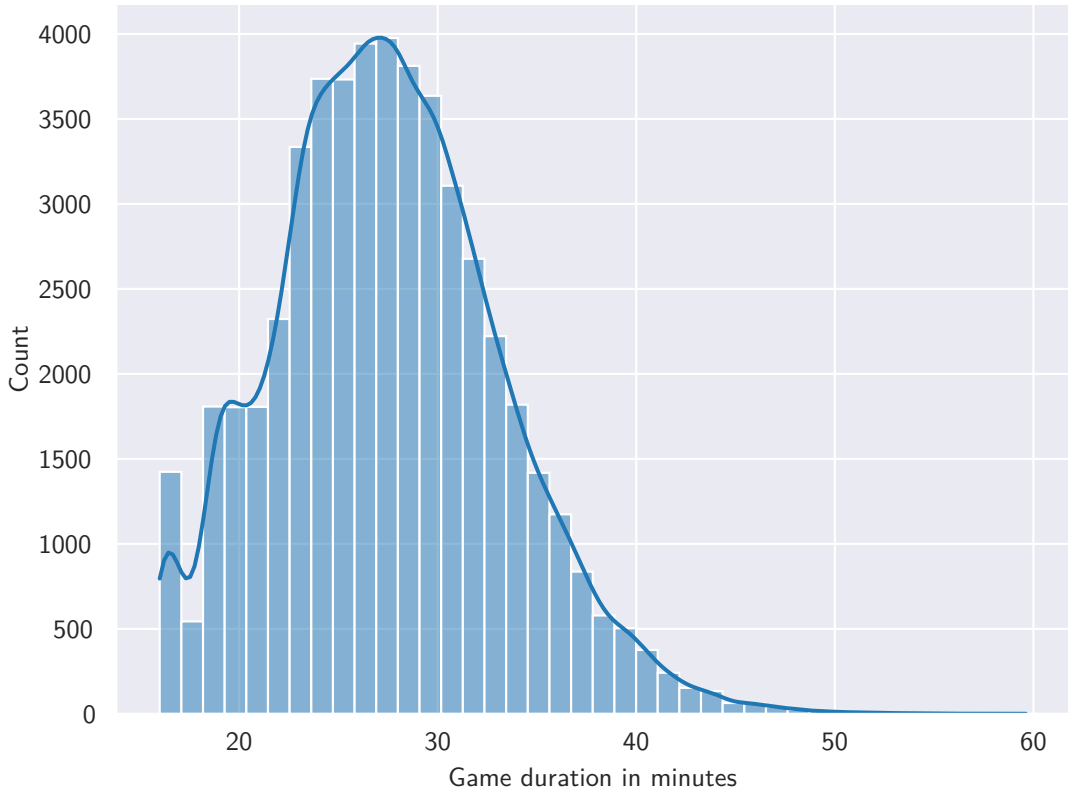

Figure 4: Distribution of game duration with its kernel density estimationd

number, are exclusively utilized for validation purposes and are excluded from the final dataset.

Player Information. This feature $\mathbf{x}_p$ is a two-dimensional vector including 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.

Champion Information. The Champion Information feature $\mathbf{x}_c$ is composed of different metrics describing the success the players have with a particular champion over all games in all ranks (e.g. win rate). Additionally, it contains more subjective information (e.g. difficulty) which is provided by Riot Games as a general guide to the champion. 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.

Player-Champion Information. This feature vector $\mathbf{x}_f$ 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. [6] found that the most pivotal feature within this category is the

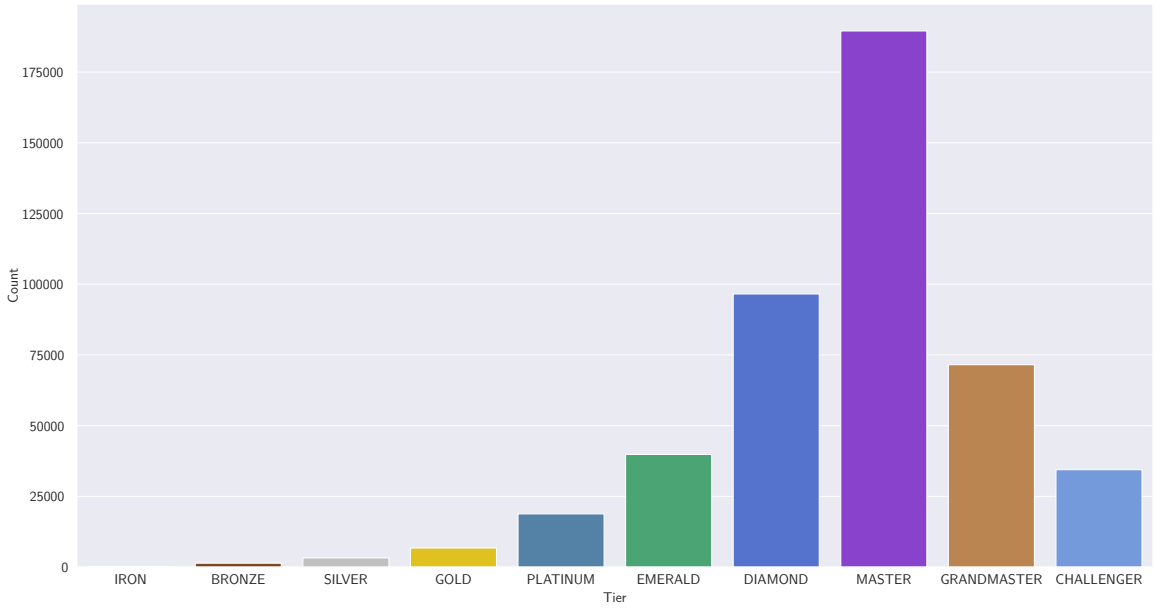

Figure 5: Distribution of ranks in the dataset

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 (e.g. average gold per match) consists of 10 features, one for each participant.

4.2.2 In-Game Dataset

The in-game dataset comprises 28,809 matches, representing a random subset extracted from the broader pre-game dataset. 473 features are used to describe the current state of the game at every minute, each forming a discrete time series. These features encompass a range of player-specific metrics, including but not limited to damage inflicted on opponents, individual champion levels, and the accumulation of gold. 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. As illustrated in Figure 6, members of the victorious team typically amass significantly more gold by the game's conclusion compared to their counterparts. This is more prominent in the later stages of the game, making the win prediction earlier more difficult. 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. Both the number of destroyed turrets and the cumulative amount of gold is tracked for each team.

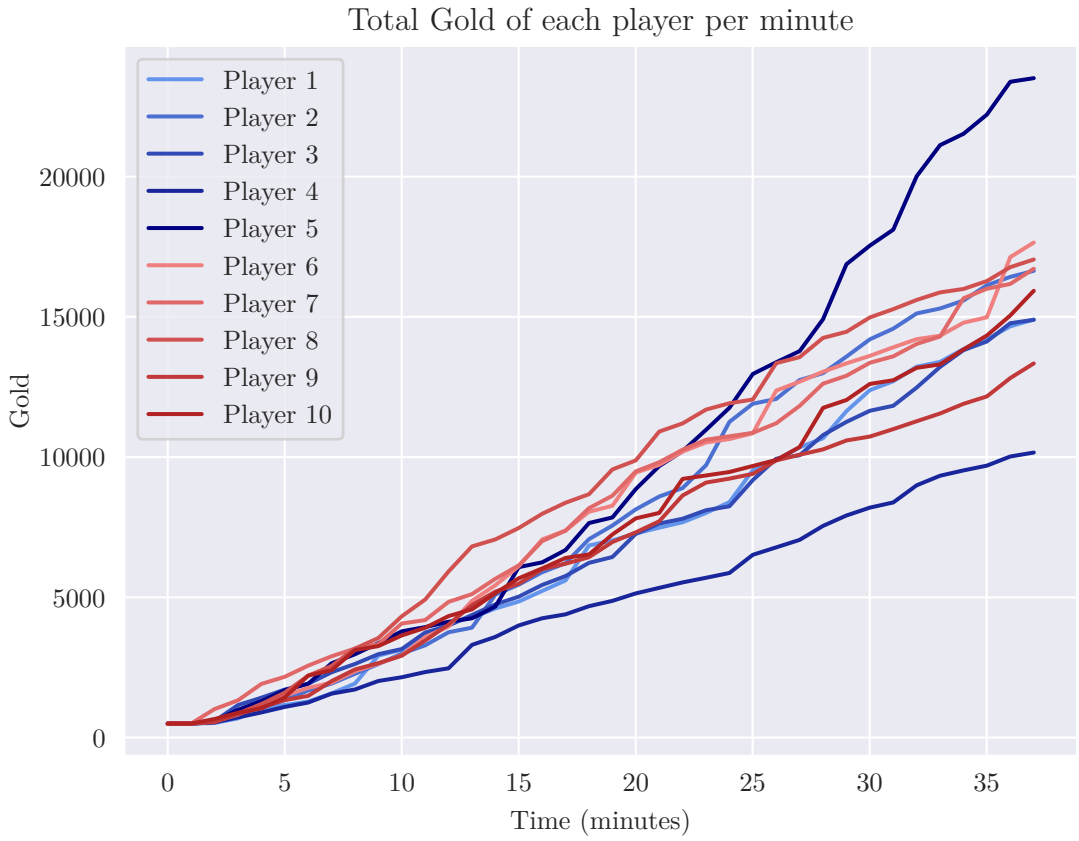

Figure 6: Total gold accumulated by each player of the course of the match, separated into teams by color.

4.3 Data Processing

Missing Values There are no missing values in the data obtained directly from Riot Games. On the rare occasion that the web scraper could not find the relevant information, missing values can occur. As this is a technical error, the missing data can be assumed as being Missing Completely At Random (MCAR) distributed [45] and thus deleted row-wise without introducing a bias. Even if this assumption should be false, due to the low number of affected samples the introduced bias would be very small.

By deleting all rows containing missing data, the dataset shrinks by 384 rows. As missing values are caused by the web scraper, the in-game dataset has no missing values.

Individual Feature Processing Pre-Game Dataset. The rank is converted from the rank-tier system into a single floating point number where the integer part denotes the tier (Master, Grandmaster, etc.) and the floating point part denotes the rank (ranges from 1-5, but only in Diamond and below). The win rate is calculated by averaging over the number of wins and losses the player has accumulated over the season. The champion tier is converted from D - S ranking into integer values and the champion number is one-hot encoded. The one-hot encoding results in two vectors of size n -one for each team- where n is the number of champions and

$$x_i = \begin{cases} 1 & \text{if a member of the team plays champion } i, \\ 0 & \text{otherwise.} \end{cases}$$

It is only possible to represent the whole team composition in one vector because no two players can pick the same champion. As the 10 features per category in $\mathbf{x}_f$ is a very fine-grained approach which greatly increases the dimensionality of the dataset, we found that a broader approach is sufficient, where statistics are averaged for each team, reducing 10 features per category to two.

Individual Feature Processing In-Game Dataset. The raw in-game dataset contains 474 features, 47 per participant and 2 columns per team containing the number of destroyed turrets and cumulative gold. Utilizing domain expertise, 9 features per participant have been deemed extraneous and subsequently excluded, effectively reducing the feature count to 384. To further streamline the dataset, a dimensionality reduction strategy is employed whereby individual player metrics are averaged across their respective teams. This method significantly decreases the total number of features, albeit at the expense of individual player variability and unique performance traits. Such a reduction inherently shifts the analytical emphasis from individual prowess to overall team dynamics. Given that LoL is intrinsically a team-oriented game, this refocused perspective is anticipated to enhance the generalizability of a predictive model as the model is trained on team-level trends rather than individual fluctuations, which can be more noisy and less predictive of outcomes.

Scaling and Partitioning The pre-game dataset is partitioned into train, validation and test set with a validation and test size of 4,000 samples $\approx 10\%$ respectively. In order to have a more comparable training dataset size, the test and validation sets from the in-game dataset contain just 1,000 matches. 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 allow proper training of the model, the data has to either be transformed into range $[1; -1]$ or standardized [46]. The standardization is performed on each feature individually by calculating the mean μ and standard deviation σ of the training set and applying the standardization $z = (x - \mu)/\sigma$ where x is the original data and z the transformed data.

5 Results and Discussion

5.1 Feature Selection Results

In order to assess the linear relationship between features and the target variable for the pre-game dataset,Pearson's Correlation Coefficient is employed. In the dataset, each feature category is split in two columns, one per team. To facilitate a comprehensive understanding of which feature category is most influential, the results are averaged across each category. Only the absolute values of the PCCs are considered, as the direction of the relationship is of no concern when assessing its strength. The ten features with the highest correlation are displayed in Table 2. It is important to note that all p-values associated with these correlations are below the significance threshold of 0.01, thereby confirming their high statistical relevance.

Gradient Boosted Trees were used to assess non-linear relationships. The model was configured with the following hyperparameters: number of estimators set to 100, maximum depth at 3, a learning rate of 0.1, and the objective function as binary logistic. The Feature Importance Score is the number of times a feature was used to split the data across all trees. GBT achieved an accuracy of 0.69.

In order to provide a clearer comparison between the PCC and GBT results, both sets of scores have been normalized into range $[0, 1]$. This normalization, as depicted in Figure 7, allows for a more intuitive visual comparison of the feature importance as assessed by the two different methods.

A key observation is that the highest PCC and Feature Importance is associated with the KDA feature. This metric, which calculates the average ratio of $\frac{\text{Kills+Assists}}{\text{Deaths}}$ achieved by a player on their champion, still demonstrates a low linear correlation with the target variable of only 0.287. Notably, the three next highest correlations are Assists, Deaths and Kills, all of which are combined to calculate the KDA. This is different to the GBT results, as the three features with the next highest score are Gold, League Points and Assists. This suggests that KDA encapsulates the essential information from Kills, Deaths and Assists, leading to GBT assigning higher importance to KDA and lower importance to these related features.

In contrast to the analysis performed by Costa et al. [6], the win rate is not the most important factor. However, the general trend of placing the highest importance on statistics of individual players instead of general champion statistics is similar.

For the in-game dataset, manual feature selection using domain knowledge was performed. Here, eight feature categories were removed due to their information being deemed either little informative about the state of the game (e.g. the bonus armor penetration) or more clearly conveyed in other features (gold earned per second is just as informative as the total gold the team has earned so far).

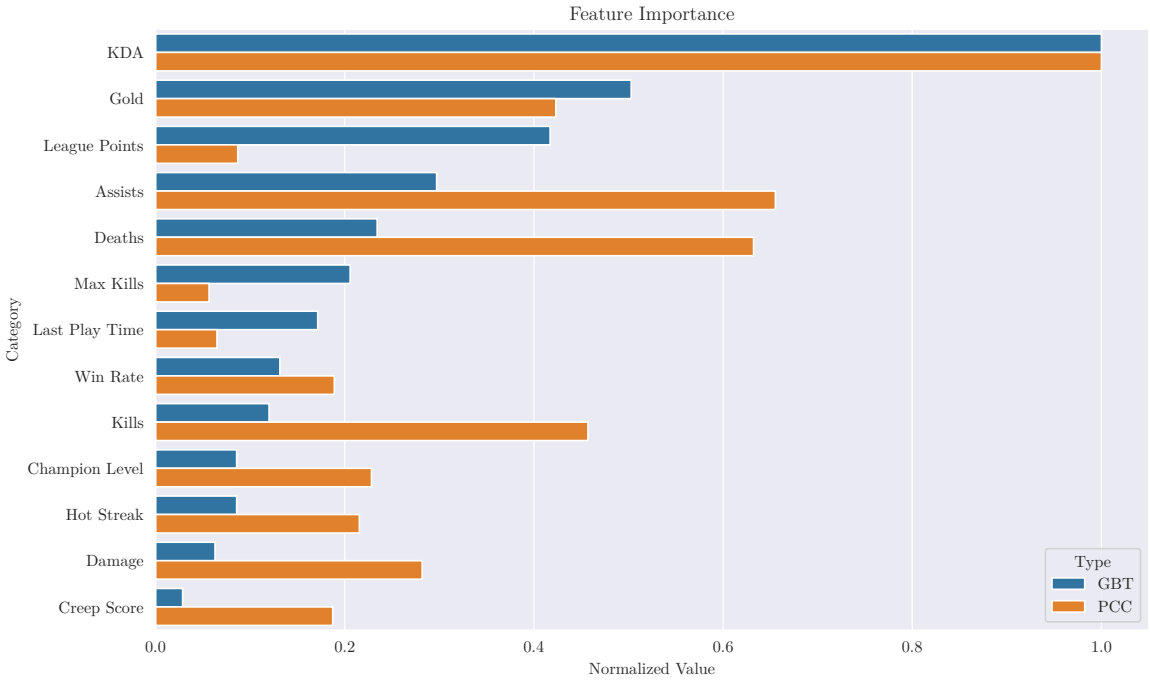

Figure 7: GBT Feature Importance Scores and PCCs normalized into range $[0, 1]$

Table 2: Gradient Boosted Tree Feature Importance Scores and Pearson's Correlation Coefficients, averaged per category

Category	GBT Importance Score	PCC
KDA	88.5	0.287
Gold	45.0	0.124
League Points	37.5	0.030
Assists	27.0	0.190
Deaths	21.5	0.183
Max Kills	19.0	0.021
Last Play Time	16.0	0.023
Winrate	12.5	0.058
Kills	11.5	0.134
Championlevel	8.5	0.069
Hot Streak	8.5	0.066
Damage	6.5	0.084
Creep Score	3.5	0.058

Table 3: Overview of the hyperparameter search for the pre-game classification

Hyperparameter	ANN	GRU	Distribution
Hidden Size	[128, 256, 512]	[64, 128, 256]	Selection
Learning Rate	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	$[1 \times 10^{-15}, 1 \times 10^{-5}]$	log uniform
Number of Layers	[2,12]	[1, 3]	discrete
Dropout Probability	[0, 0.3]	[0, 0.3]	discrete
Activation Function	[ReLU, ELU]	[ReLU]	Selection
Model	[ANN]	[RNN, GRU]	Selection

5.2 Hyperparameter Optimization

The following settings were determined manually and fixed prior to the hyperparameter optimization.

ANN Parameters The ANN model architecture includes a series of fully connected layers, each followed by 1-dimensional batch normalization. The exact number of layers is one of the primary parameters determined by the hyperparameter optimization. The batchnorm is used to stabilize the learning process, as initial experiments have shown highly fluctuating training loss. The activation function for each neuron is the Expontential Linear Unit, as the experiments and Do et al. [5] have shown some improvement with it. Binary Cross-Entropy Loss was used as the loss function and adam optimizer was employed due to its adaptive learning rate capabilities, which can help mitigate issues with unstable gradients [47]. In order to avoid overfitting, the training is stopped when the validation accuracy does not improve for 30 epochs.

GRU Parameters The GRU model uses the following fixed parameters: One recurrent layer is determined to be sufficient with a number of fully connected, non recurrent layers following. Similar to the ANN, Binary CEL and adam optimization is used. The early stopping patience is set to 100 epochs, as the training converges much slower than the ANN training.

Focus and Metrics The primary focus of our hyperparameter optimization was twofold: to determine the optimal depth of the neural network and to identify the most effective learning rate. The evaluation of model quality centered on validation accuracy, deemed a suitable metric given the near-perfect balance of the dataset and the equivalent cost of false positives and negatives in predictions.

Search Strategy The search space, shown in Figure 3 is randomly searched. This method allows for a comprehensive exploration of potential configurations, striking a balance between thoroughness and computational feasibility [48].

Table 4: Final hyperparameters for ANN and GRU

Parameter	ANN	GRU
Hidden Size	256	128
Learning Rate	1.3×10^{-7}	1×10^{-10}
Number of Layers	6	2
Dropout Probability	0	0
Activation Function	ELU	ReLU

5.3 Classification Results

ANN Model Performance The ANN has demonstrated a test accuracy of 0.710 and a ROC-AUC Score of 0.787 (see Figure 8). The confusion matrix can be seen in Figure 9. This performance aligns closely with previous findings in Chapter 2, suggesting that while professional games may be predicted more accurately, lower-rated games are more unpredictable and challenging. This is consisten with White et al.'s [4] findings, who, despite a more complex methodology and larger dataset, achieved similar accuracy in predicting lower-ranked games. As the most important features are related to player statistics, it can be assumed that the model generalizes fairly well over multiple game versions. Nevertheless, the model will lose predictive power after large changes of the game, which underscores the necessity of monitoring and frequent retraining.

GRU Model Performance The GRU model achieved an accuracy of 0.735, marginally surpassing the pre-game ANN model. This result was somewhat unexpected, as in-game predictions generally have the advantage of real-time data. This suggests that pre-game data is not significantly less predictive than early real-time data, especially if only the real-time data is used.

Nevertheless, it shows that information obtained from a later state in the game can increase predicitive power.

Silva et al. [7] have employed an RNN with an accuracy of 0.752 at the 15th minute and found that GRU did not perform as well. They hypothesized that the underperformance of GRU compared to a regular RNN might be due to a lack of data. With a much larger dataset, GRU did in fact perform slightly better than a vanilla RNN, but the difference is marginal, as the best performing RNN achieved an accuracy of 0.730. A potential improvement lies in the fact that the GRU model did not utilize any pre-game data, which have been shown to predict the winner with reasonable accuracy. A combination of both data types might yield a much better predictor. These results also show that a simple increase in data does not always lead to better predictions. Here, it can be assumed that the data does not encompass all relevant information needed to make a highly accurate prediction or that a significant portion of the game's outcome might remain beyond the predictive capacity of algorithms due to the intrinsic randomness and complexity of human-led interactions.

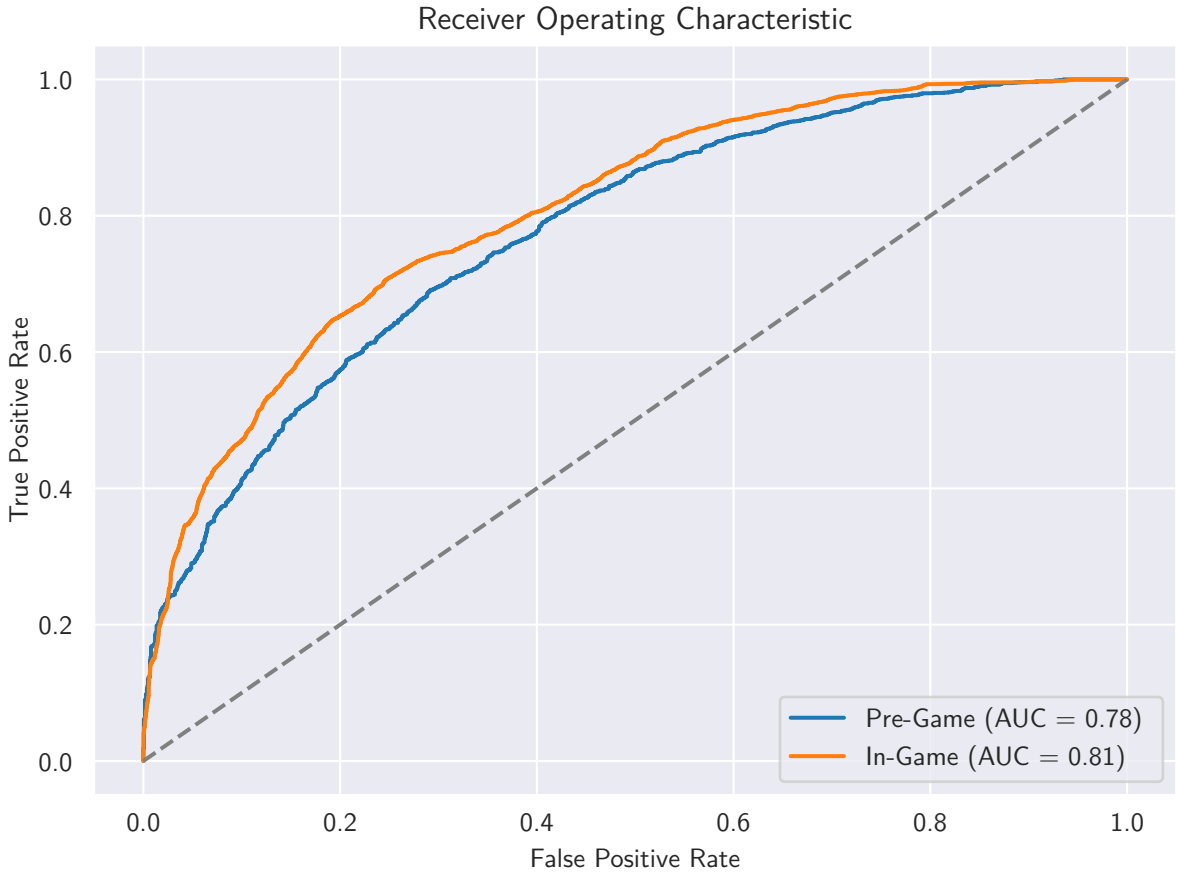

Figure 8: ROC Curve from both Models

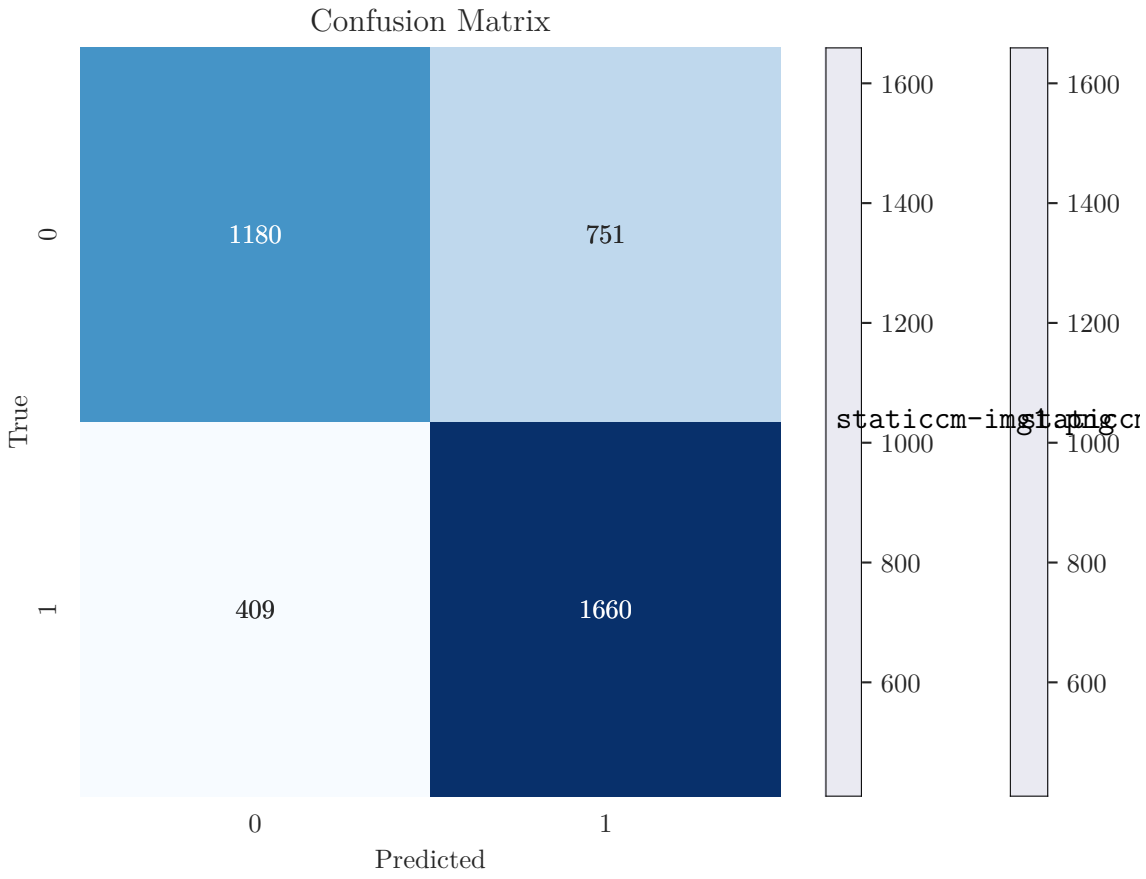

Figure 9: Confusion matrix of pre-game classification results

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. 2020.

[4] A. White and D. M. Romano. Scalable Psychological Momentum Forecasting in Esports. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020.

[5] 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. 2021.

[6] 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.

[7] 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.

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

[9] 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. 2023.

[10] 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.

[11] 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.

[12] L. Lin. League of Legends Match Outcome Prediction. *Comput. Sci. Dept., Univ. Stanford, Stanford, CA, USA, Rep*, 2016.

[13] D.-H. Kim, C. Lee, and K.-S. Chung. A Confidence-Calibrated MOBA Game Winner Predictor. In: *2020 IEEE Conference on Games (CoG)*, 622–625, Aug. 2020.

[14] 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.

[15] 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.

[16] 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.

[17] *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. 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 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*. Feb. 2019. 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 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*. Sept. 2014. arXiv: 1406.1078 [cs, stat]. 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. 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*. Dec. 2014. arXiv: 1412.3555 [cs]. 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. *Journal of machine learning research* 3, 1157–1182, 2003.

[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. 2016.

[39] R. Games. *Master, Grandmaster, and Challenger: The Apex Tiers*. Aug. 2023. URL: https://support-leagueoflegends.riotgames.com/hc/en-us/articles/4405776545427-Master-Grandmaster-and-Challenger-The-Apex-Tiers (visited on 01/01/2024).

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

[41] R. Games. */Dev: Balance Framework Update - League of Legends*. June 2020. URL: https://www.leagueoflegends.com/news/dev/dev-balance-framework-update/ (visited on 11/09/2023).

[42] *Riot Developer Portal*. URL: `https://developer.riotgames.com/` (visited on 12/19/2023).

[43] *U GG: The Best League of Legends Builds LoL Build Champion Probuilds LoL Runes Tier List Counters Guides*. URL: `https://u.gg` (visited on 12/15/2023).

[44] 2023 *League of Legends* World Championship. *Wikipedia*, Dec. 2023.

[45] A. C. Acock. Working With Missing Values. *Journal of Marriage and Family* 67(4), 1012–1028, 2005.

[46] M. Shanker, M. Y. Hu, and M. S. Hung. Effect of Data Standardization on Neural Network Training. *Omega* 24(4), 385–397, Aug. 1996.

[47] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. Jan. 2017. arXiv: `1412.6980 [cs]`. URL: `http://arxiv.org/abs/1412.6980` (visited on 01/16/2024).

[48] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization.

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

MCAR Missing Completely At Random

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