OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Comparative Analysis of Predictive Performance: Neural Networks vs. GRU in League of Legends Match Outcome Prediction

Bachelorarbeit
von

Moritz Palm
Matrikelnummer: 1234567

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

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

Abgabedatum: 2. 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 2. Dezember 2023

Moritz Palm

Inhaltsverzeichnis

1 Introduction **1**

2 Background **3**
2.1 League of Legends 3
2.2 Neural Networks 4
2.3 Recurrent Neural Networks 5
2.4 GRU 7
2.5 Related work 7

3 Data **9**
3.1 Data Collection 9
3.2 Dataset Properties 10
3.3 Data Processing 10
3.3.1 Pre-Game Dataset 10
3.3.2 In-Game Dataset 13

4 Experiments **15**
4.1 Feature Selection 15
4.1.1 Pearsons' correlation coefficient 15
4.1.2 Gradient Boosted Trees 15

5 Results **17**

6 Discussion **19**

7 Conclusion **21**

Bibliography **23**

List of Figures **25**

List of Tables **27**

List of Abbreviations **29**

1 Introduction

esports is highly relevant due to it being a huge and strongly growing market. 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 predicition 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 hyperscaling heroes can come back and win

2 Background

2.1 League of Legends

League of Legends (LoL) is a Multiplayer Online Battle Arena (MOBA) game developed by Riot Games. MOBA games, a subgenre of real-time strategy games, typically feature two teams, each consisting of five players known as 'summoners'. These players compete against each other, with each controlling a unique character, referred to as a 'champion' [1]. LoL is among the most popular video games in this genre, drawing millions of players worldwide and a substantial viewership in professional esports tournaments [CITATION NEEDED].

While most MOBA games share similar basic gameplay elements and map layouts, they differ in aspects such as available champions, abilities, and graphical styles. Given its prominence in the MOBA genre, this thesis will primarily focus on League of Legends. In LoL, players begin each match by selecting a champion from a roster of 165, each with distinct abilities and characteristics. The game map is divided into two bases connected by three lanes, with each base housing a critical structure called the 'nexus', defended by turrets. The primary objective is to destroy the opposing team's nexus.

The game's map also features a jungle area, which contains neutral monsters and two significant creatures, Baron Nashor and the Dragon. These monsters provide team-wide benefits upon defeat. The game's strategic complexity is heightened by the need for players to earn gold and experience points (xp) by defeating minions, neutral monsters, or opposing champions. This currency is used to purchase items and level up, enhancing a champion's capabilities.

Player roles are traditionally distributed with one player in the top lane, one in the mid lane, two in the bottom lane, and one in the jungle. This distribution allows for strategic diversity and specialization. Each year, LoL enters a new 'season', introducing significant changes. Additionally, Riot Games releases bi-weekly patches to adjust champion balance, potentially altering the prevailing game strategies, or 'meta'.

Champion selection is a critical aspect of LoL gameplay. Players must consider various factors such as team composition, damage types, assigned roles, and personal proficiency with specific champions. Theoretically, the number of possible champion combinations in a game is $\binom{165}{10} = \frac{165!}{10! \cdot (165-10)!}$, highlighting the game's strategic depth.

To assess individual player skill, LoL employs a modified Elo rating system. This system is crucial for matchmaking, ensuring players are paired with and against others of similar skill levels.

2.2 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. [2]. 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 [3], 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:

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

where $\mathbf{a}^{(l)}$ represents the activation of layer l, $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ is the bias vector. The output layer L produces the network's prediction $\hat{y}$.

To approximate any measurable function, an ANN requires at least one hidden layer [4]. 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:

$$E_N = \frac{1}{N} \sum_{k=1}^{N} (y_k - \hat{y}_k)^2, \tag{4}$$

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

$$E_N = -\frac{1}{N} \sum_{k=1}^{N} \left(y_k \ln \hat{y}_k + (1 - y_k) \ln (1 - \hat{y}_k) \right), \tag{5}$$

where N is the number of samples, y_k is the true label, and $\hat{y}_i$ or $\hat{y}_k$ is the predicted value or probability.

Backpropagation 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:

$$\frac{\partial E}{\partial \mathbf{W}^{(l)}} = \frac{\partial E}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}}, \tag{6}$$

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.

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.

2.3 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 [5].

The concept of a fully connected RNN was first proposed by Elman [6]. RNNs maintain a 'state vector' in their hidden units, which implicitly contains information extracted from all past elements of the sequence [5]. The output $\hat{y}_t$ of an RNN at time step t can be calculated as:

$$\hat{y}_t = \tanh(Vh_t + b_o), \tag{7}$$

where V is the weight matrix and b_o is the bias vector associated with the cell output. The hidden state h_t at each time step t is updated as follows:

$$h_t = \begin{cases} 0, & \text{if } t = 0, \\ \sigma(Wx_t + Uh_{t-1} + b_h), & \text{otherwise}, \end{cases} \tag{8}$$

where σ is the sigmoid activation function, U and b_h are the weight matrix and bias vector for the hidden state, and W is the weight matrix for the input. In a single-layer RNN, one hidden weight matrix W_h is shared across all timesteps.

BPTT unfolds the RNN across time steps (see figure 2) and applies the backpropagation algorithm. The gradients of the loss function with respect to the weights are calculated for each time step. For a given time step t, the gradient of the loss E with respect to the weights W is:

$$\frac{\partial E}{\partial W} = \sum_{\tau=1}^{t} \frac{\partial E}{\partial \hat{y}_\tau} \cdot \frac{\partial \hat{y}_\tau}{\partial h_\tau} \cdot \frac{\partial h_\tau}{\partial W}, \tag{9}$$

where $\frac{\partial E}{\partial \hat{y}_\tau}$ is the gradient of the loss with respect to the output at time τ, $\frac{\partial \hat{y}_\tau}{\partial h_\tau}$ is the gradient of the output with respect to the hidden state, and $\frac{\partial h_\tau}{\partial W}$ is the gradient of the hidden state with respect to the weights. This process is repeated for all weights and biases in the network, allowing the RNN to learn from sequences by adjusting its parameters based on the temporal context.

However, as demonstrated by Bengio et al. [7], RNNs face challenges with exploding or vanishing gradients, particularly in long sequences. This issue hinders their ability to learn long-range dependencies [8].

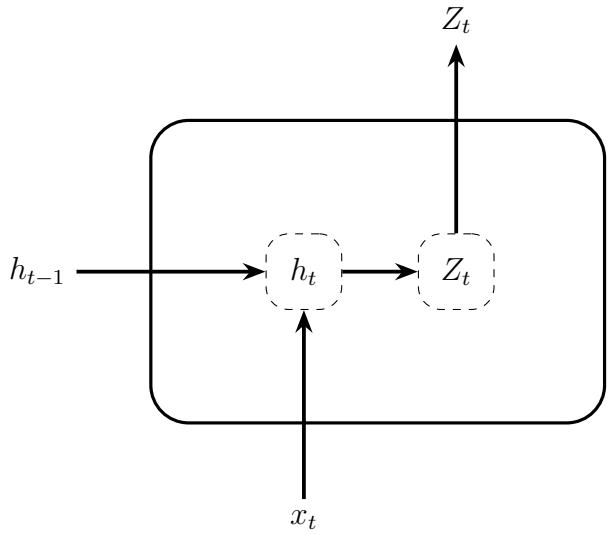

Figure 1: Elman Recurrent Unit

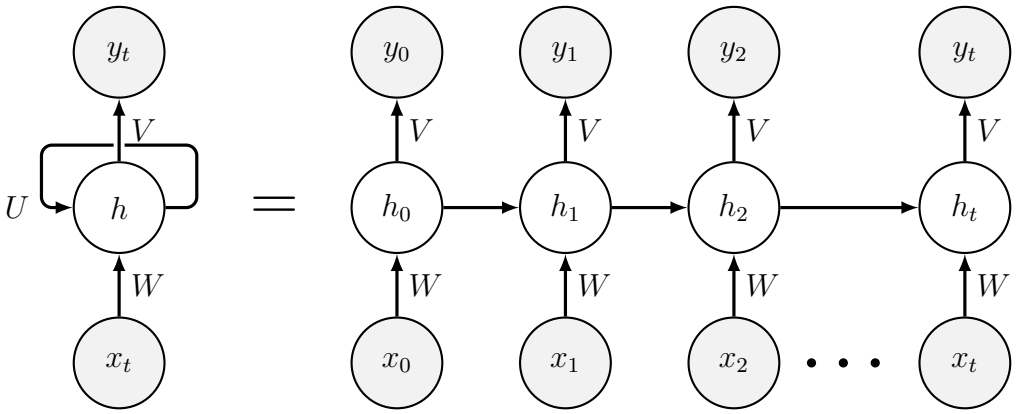

Figure 2: Unrolling of an Recurrent Neural Network (RNN) over time

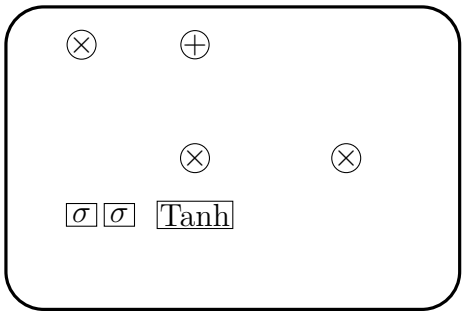

Figure 3: Gated Recurrent Unit

2.4 GRU

In order to overcome the exploding/vanishing gradient problem of vanilla Recurrent Neural Networks (RNNs), gated networks like the Long Short-Term Memory (LSTM) [9] and Gated Recurrent Unit (GRU) [10] have been developed [11]. As they introduce an increased number of parameters compared to traditional RNNs, gated networks like the LSTM and GRU demand greater computational power [12]. Compared to the LSTM network, GRU reduces the number of gate networks to two, thus being simpler to implement and compute [10]. Chung et al. even found that GRU is at least comparable to LSTM most of the time [13]. The gates control the activation of each hidden unit. The reset gate is calculated by

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{10}$$

and the update gate z_j by

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{11}$$

[12]. 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 [13]:

$$h_t = (1 - z_t) \odot h_t + z_t \odot \tilde{h}_t \tag{12}$$

with

$$\tilde{h}_t = g(W_h x_t + U_h(r_t \odot h_{t-1} + b_h) \tag{13}$$

. In equations 12 and 13 $\odot$ denotes the elementwise (Hadamard) multiplication and the function g in equation 13 is the activation function.

2.5 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 3.1). 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. [14] trained a 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 [15] 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 [16].

3 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.

3.1 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' [17], 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 [18].

Riot Games API The primary source of data stemmed from the Riot Games API, 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 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. The amalgamation of Riot Games API data and web-scraped data was meticulously organized and stored in a Database. This central repository served as the foundational storehouse from which distinct datasets were constructed, ensuring an organized and structured approach to data management.

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 $20,513$ 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.

3.2 Dataset Properties

The matches collected have the following properties: 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 26.88 minutes.

3.3 Data Processing

3.3.1 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.

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

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 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

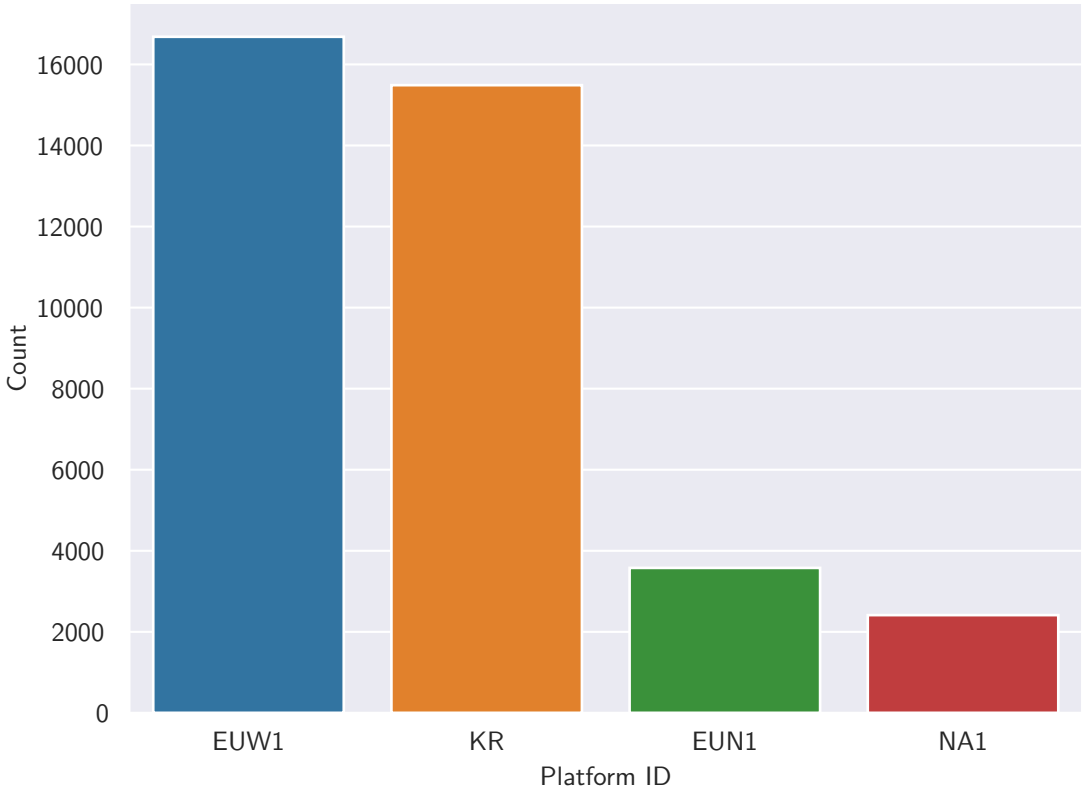

Figure 4: Region distribution of all matches in the dataset

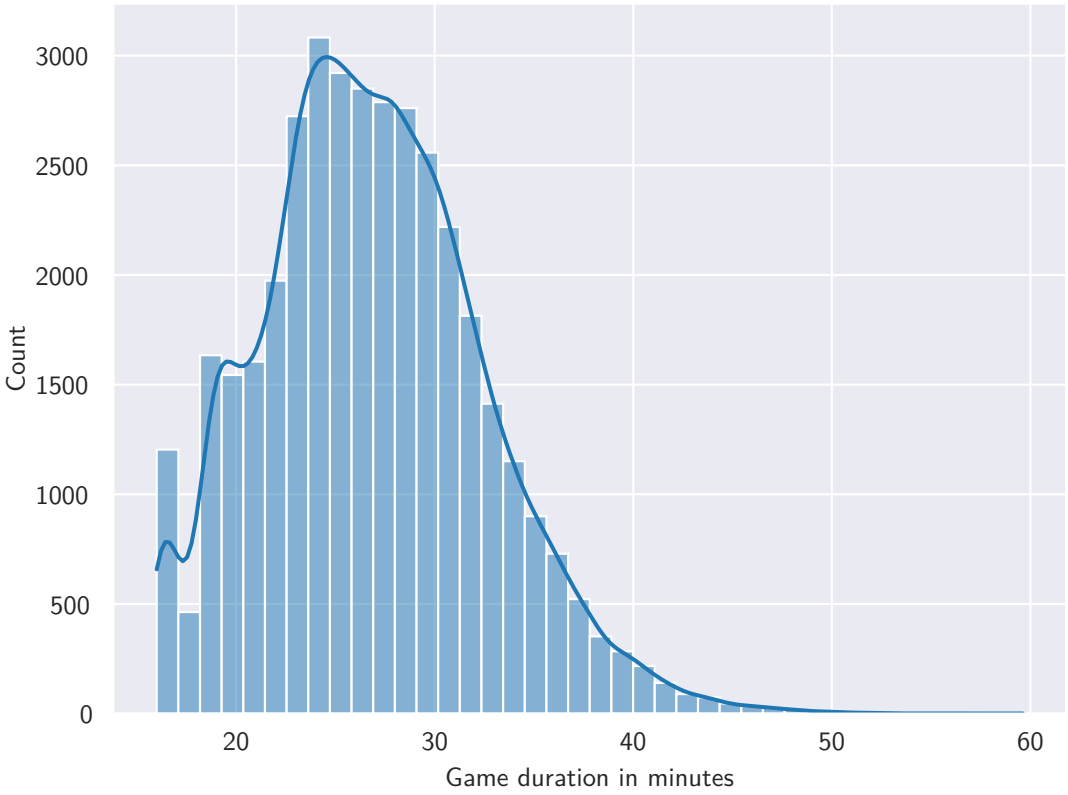

Figure 5: Distribution of game duration with its kernel density estimation. The spike at 16 minutes is explained by the fact that this is the earliest possible surrender time.

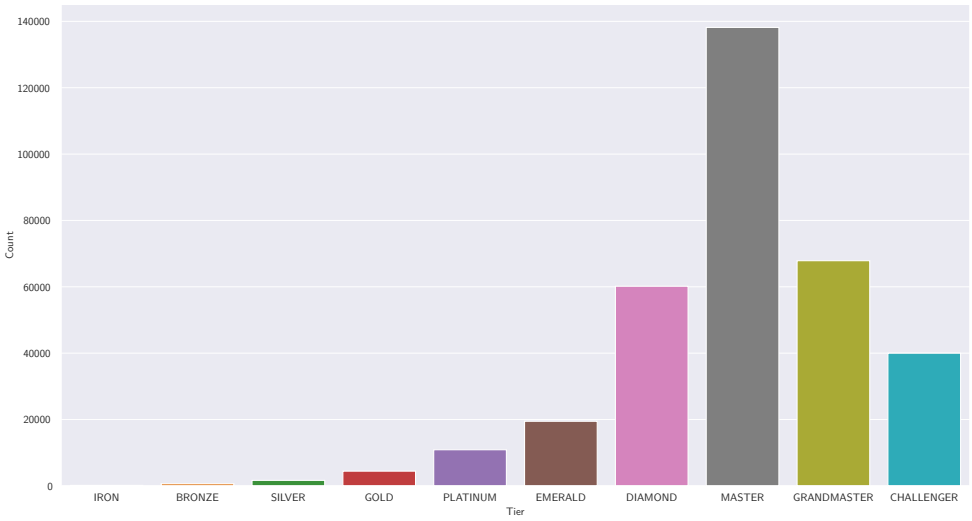

Figure 6: Distribution of ranks in the dataset

13. Costa et al. [19] 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.

3.3.2 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 analyzing gold trends. As illustrated in Figure 7, 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.

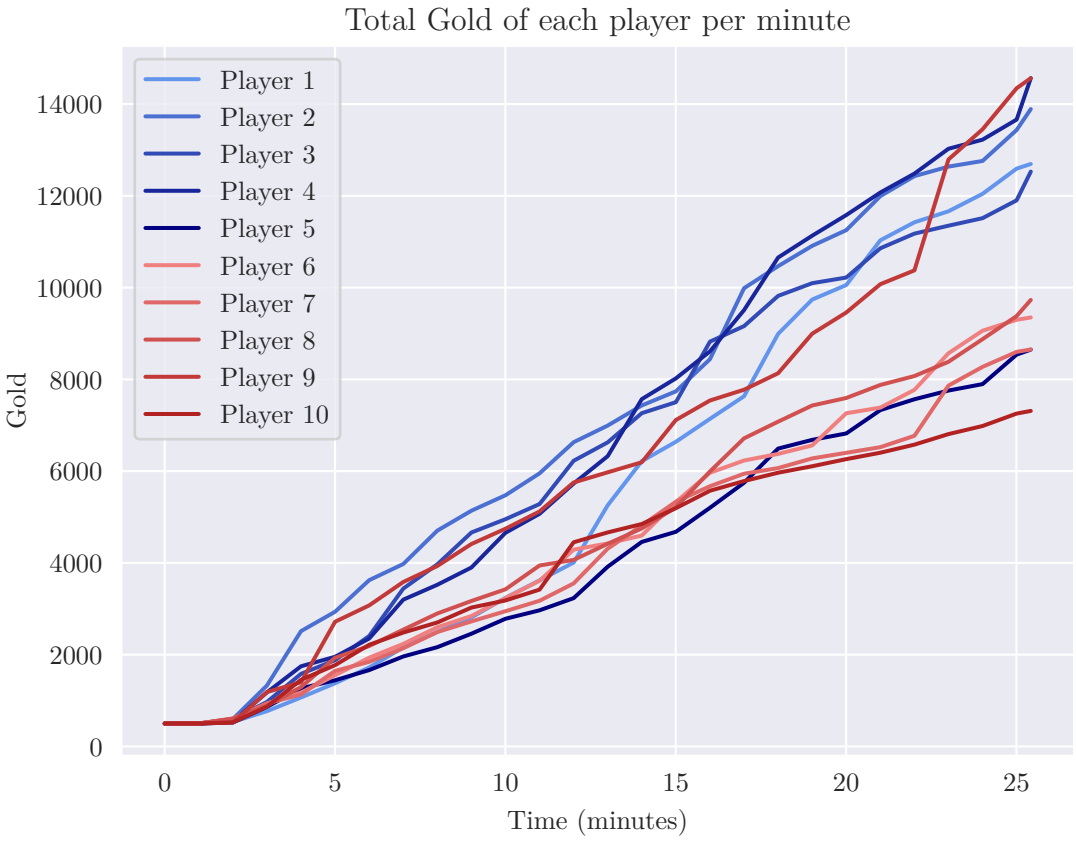

Figure 7: Total gold accumulated by each player of the course of the match, separated into teams by color. A clear separation between the blue and red team is noticeable especially at the later stages of the game.

4 Experiments

4.1 Feature Selection

Both datasets initially have a very high dimensionality, which can lead to the prevalence of noisy or irrelevant data [20]. In order to facilitate the learning of the ANN, all such data should be removed. This process of finding the best subset of features used for training is called Feature Selection. Different methods of selecting a subset of features have been tried:

- Pearson's correlation coefficient
- Gradient Boosted Trees

4.1.1 Pearsons' correlation coefficient

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 $\mathbf{y}$ is the target value vector, viewed as a realization of the random variable Y [21]. The Pearson correlation coefficient is employed to quantify the linear correlation between these two random variables. It is defined by the formula:

$$R(i) = \frac{\text{cov}(X_i, Y)}{\sqrt{\text{var}(X_i) \cdot \text{var}(Y)}}, \tag{14}$$

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 [22]. In order to ascertain the significance of the test results, a hypothesis test is performed, with the null hypothesis being that there is no correlation between the feature and the target.

The results of the test are found in table 1. 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.

4.1.2 Gradient Boosted Trees

Gradient Boosted Trees (GBTs) like all tree-based methods have an embedded method of selecting the most important feature. There are different types of importance, such as the average or total gain across all splits the feature is used in. The simplest definition

Feature	Correlation coefficient	GBT Feature Importance
participant_9_champion_kda	0.148	
participant_5_champion_kda	-0.148	
participant_4_champion_kda	-0.137	
participant_7_champion_kda	0.136	
participant_3_champion_kda	-0.133	
participant_10_champion_kda	0.132	
participant_2_champion_kda	-0.131	
participant_8_champion_kda	0.129	

Table 1: Pearson's correlation coefficient for the 15 features with the highest absolute correlation

is the 'weight', defined as the number of times a feature is used to split the data across all trees [**chenXGBoostScalableTree2016**]. 30 features have not been used in any split and thus have a feature importance of 0.

5 Results

6 Discussion

7 Conclusion

Bibliography

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

[2] 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.

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

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

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

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

[7] 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.

[8] 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).

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

[10] 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).

[11] 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.

[12] 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.

[13] 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).

[14] 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.

[15] 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).

[16] 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.

[17] 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).

[18] 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).

[19] 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.

[20] B. Venkatesh and J. Anuradha. A Review of Feature Selection and Its Methods. *Cybernetics and Information Technologies* 19(1), 3–26, Mar. 1, 2019.

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

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

List of Figures

1 Elman Recurrent Unit 6
2 Unrolling of an RNN over time 6
3 Gated Recurrent Unit 7

4 Region distribution of all matches in the dataset 11
5 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. 12
6 Distribution of ranks in the dataset 12
7 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. 14

List of Tables

1 Pearson's correlation coefficient for the 15 features with the highest absolute correlation 16

List of Abbreviations

ANN Artificial Neural Network

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

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

xp experience points