FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Embedding League of Legends player styles for clustering and outlier detection
Student:	Nikolay Tsoy
Supervisor:	Ing. Jan Drchal, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

The goal is to develop methods of clustering players in the League of Legends online game. Experiment with outlier detection methods applied to embedding sequences derived from game history statistics. A significant part lies in the design of embedding extraction algorithms.

1) Create a large enough dataset by pulling data using the Riot official API.
2) Familiarize yourself with state-of-the-art embedding methods for multidimensional sequences.
3) Apply existing (or design your own) methods to extract the game history of League of Legends player embeddings.
4) Cluster players based on their embeddings and detect outliers. Discuss results with respect to different playing styles.

[1] Kim, Hwa Jong, Seong Eun Hong, and Kyung Jin Cha. "seq2vec: analyzing sequential data using multi-rank embedding vectors." Electronic Commerce Research and Applications 43 (2020): 101003.
[2] You, Quanzeng, Zhengyou Zhang, and Jiebo Luo. "End-to-end convolutional semantic embeddings." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.
[3] Koryťák, Martin. "Anomaly Detection Methods for Log Files." MS thesis. České vysoké učení technické v Praze, 2021.

Electronically approved by Ing. Karel Klouda, Ph.D. on 10 February 2022 in Prague.

[4] Chalapathy, Raghavendra, and Sanjay Chawla. "Deep learning for anomaly detection: A survey."; arXiv preprint arXiv:1901.03407, 2019.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Embedding League of Legends player styles for clustering and outlier detection

Nikolay Tsoy

Department of Applied Mathematics
Supervisor: Ing. Jan Drchal Ph.D.

May 10, 2022

Acknowledgements

I want to express my deepest appreciation to my supervisor Ing. Jan Drchal Ph.D., who encouraged me throughout this project. Thanks to his valuable advice, profound belief in my work, and patience, that cannot be underestimated.

I also wish to thank my family, friends, and everyone involved. Thank you for your unwavering support and practical suggestions.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 10, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tsoy, Nikolay. *Embedding League of Legends player styles for clustering and outlier detection.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Oblast online her je předmětem značného zájmu a množství dat neustále roste. Tyto informace můžeme využít k lepšímu modelování výkonu hráčů a pochopení jejich herních stylů a dovedností. Tato práce analyzuje herní data z League of Legends, jedné z nejpopulárnějších kompetitivních online her typu MOBA. Experimenty jsme prováděli na ručně shromážděné sadě dat, která obsahuje informace o 141 300 zápasech s 202 příznaky pro 1413 unikátních hráčů (100 posledních zápasů pro každého hráče). Cílem této práce bylo představit přístup pro generování reprezentativních hráčských embeddingů. Za tímto účelem jsme natrénovali dva modely neuronových sítí, Multilayer Autoencoder a Stacked LSTM Autoencoder. Porovnali jsme embeddingy založené na různých technikách předzpracování dat a shlukovali jsme embeddingy pomocí algoritmu K-means. Úspěšně se nám podařilo rozdělit hráče do odlišných shluků. K analýze vytvořených shluků jsme použili metody významnosti příznaků. Kromě toho jsme použili Unsupervised Anomaly Detection model k předpovědi následujících statistik zápasů. Přínosy této práce mají potenciál být využity ve službách analýzy online her k vytváření otisků hráčů, předpovídání výsledků her a odhalování podezřelého chování.

Klíčová slova Strojové učení, Modelování hráčů, League of Legends, LSTM Autoencoder, Embeddingy, Detekce odlehlých hodnot

Abstract

The online games area is attracting considerable interest, and the quantity of data is constantly growing. We can utilize this information to better model players' performance and understand their playing styles and skills. This work analyzes game data from League of Legends, one of the most popular competitive MOBA online games. We performed our experiments on the hand-collected dataset that contains information about 141 300 matches with 202 features for 1413 unique players (100 most recent matches for every player). The goal of this work was to present an approach for generating representative player embeddings. For this purpose, we trained two neural network models, Multilayer Autoencoder and Stacked LSTM Autoencoder. We compared the embeddings based on different data preprocessing techniques and clusterized the embeddings using the K-means algorithm. We successfully managed to split players into distinct clusters. We employed feature importance methods to analyze produced clusters. Moreover, we used the Unsupervised Anomaly Detection model to predict the following match statistics. The contributions of this work have the potential to be used in online game analysis services to create player fingerprints, predict game outcomes, and detect suspicious behavior.

Keywords Machine learning, Player modeling, League of Legends, LSTM Autoencoder, Embeddings, Outlier detection

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Understanding unique playstyles through data is an important area of research in machine learning. The increasing popularity of online games and the vast amount of data that are now available have led to the possibility of using such data to model players' performance. Most of the research in this area uses supervised methods to predict game outcomes based on player experience (Do et al., 2021) and collected in-game statistics (S.-K. Lee et al., 2020). It is also worth noting that, in many games, players start making choices before the match begins. (Costa et al., 2021) analyze a pick/ban phase features for victory prediction. (H. Lee et al., 2022) were able to provide personalized recommendations when choosing a hero in a multiplayer online battle arena (MOBA) game.

While there are a lot of computer games that we could choose to analyze, League of Legends (LoL), the MOBA game, seems to be one of the most popular with a convenient official API (Riot Games, 2022). At the start of each match, competitors are separated into two teams facing each other, and each player selects a champion they will play for. The game's objective is to destroy the enemy base, known as the nexus. The system constantly aggregates in-game statistics about each player's performance during a match, including damage dealt to different targets (e.g., champions, buildings), damage taken, experience collected, gold spent, and others. Individual players differ in their skill level, adaptivity to different situations, decision-making, and communication skills. It is fair to say that their performance is influenced by hardware and internet connection, so even a tiny lag during the most critical moment can dramatically change the match outcome.

Despite the possibility of using machine learning, current services that process game statistics and report players' performance use purely statisti-

cal methods. This applies both to the in-game mechanisms shown in Figure 1.1 and to third-party services such as Mobalytics[1] shown in Figure 1.2.

Figure 1.1: In-game stats page.

Figure 1.2: Mobalytics player's page.

[1] `https://app.mobalytics.gg/lol`

Although these services provide helpful information for players, machine learning approaches could quickly improve their reports' quality by identifying competitors' hidden strengths, weaknesses, and playstyles. Such a toolkit can be used for game outcome prediction, suspicious behavior detection, tracking playstyle changes, player clustering, and finding similar (in terms of playstyle) players.

We set the following goals for this thesis:

- Describe the problem of players modeling, clustering, and detecting outliers in online games.
- Create a dataset by pulling data using the Riot Games API.
- Develop a preprocessing pipeline for pulled data.
- Choose appropriate state-of-the-art embedding methods for multidimensional sequences with different approaches.
- Evaluate selected methods and discuss the results concerning different playing styles.

In the following sections, we introduce the area of multidimensional sequence modeling and encoding, analyze previous work and propose our approach to solve the matter. Finally, we show and discuss achieved results.

Chapter 2

Theoretical background

This chapter briefly describes used neural network architectures, clustering, and feature importance algorithms.

2.1 Recurrent neural networks

A recurrent neural network (RNN) (David E. Rumelhart et al., 1987) is a type of artificial neural network that can model temporal dynamic behavior by creating a directed or undirected graph of connections between nodes over a sequence. This internal state, or memory, allows the network to process a variable-length sequence of inputs. So RNNs have loops since the output of the next step depends on the previous output. Unrolling process of such a network is shown in Figure 2.1.

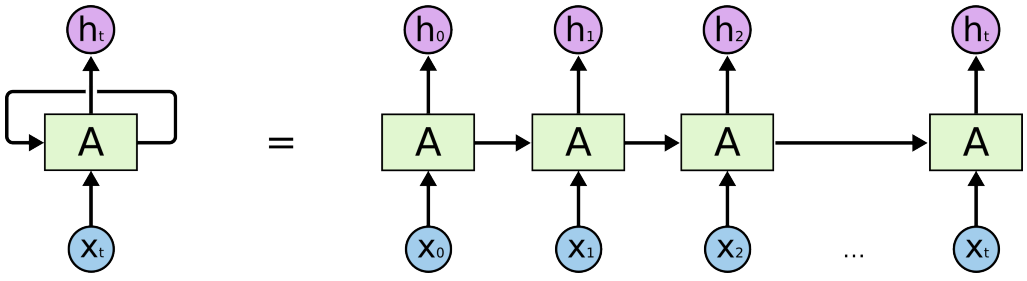

Figure 2.1: Unrolled RNN (Olah, 2015).

Where x_t is the input tensor, h_t is the output tensor (aka hidden state) and A represents the network as a function.

One of the representatives of this class of neural networks, long short-term memory (LSTM) (Hochreiter et al., 1997), is used in this thesis. LSTM is designed to overcome the vanishing and exploding gradient problems that are common in conventional RNNs (Greff et al., 2017). LSTM scheme is shown in Figure 2.2.

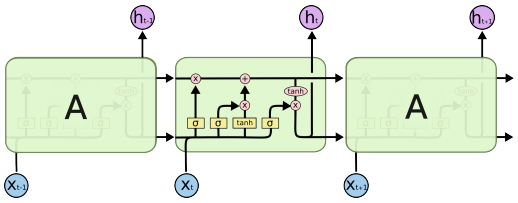

Figure 2.2: LSTM (Olah, 2015).

Where σ and $tanh$ are activation functions. All weights are shared between steps.

2.2 Embeddings

2.2.1 Autoencoder

An autoencoder (D. E. Rumelhart et al., 1986) is an artificial neural network used to encode input data into a lower dimension representation efficiently. It consists of two separate networks, an encoder, and a decoder. The encoder produces a latent representation (aka embedding, context, or code) of the input, whereas the decoder reconstructs input from the embedding. Because Autoencoder does not need labels, it is a member of the unsupervised machine learning models' family. The scheme of an autoencoder is shown in Figure 2.3

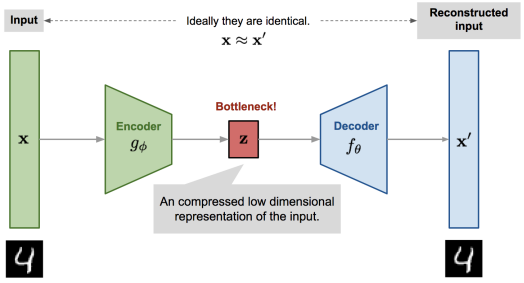

Figure 2.3: Autoencoder (Weng, 2018).

Where x is the input, x' is the reconstructed input, g_ϕ is the encoder function, f_θ is the decoder function, z is the embedding.

2.2.2 LSTM Autoencoder

An LSTM Autoencoder is a specific version of an autoencoder where an encoder and a decoder are LSTM networks. The output from the encoder's last step is used to embed the whole sequence of inputs. The decoder tries to replicate inputs given the embedding. The scheme of an LSTM Autoencoder is shown in Figure 2.4

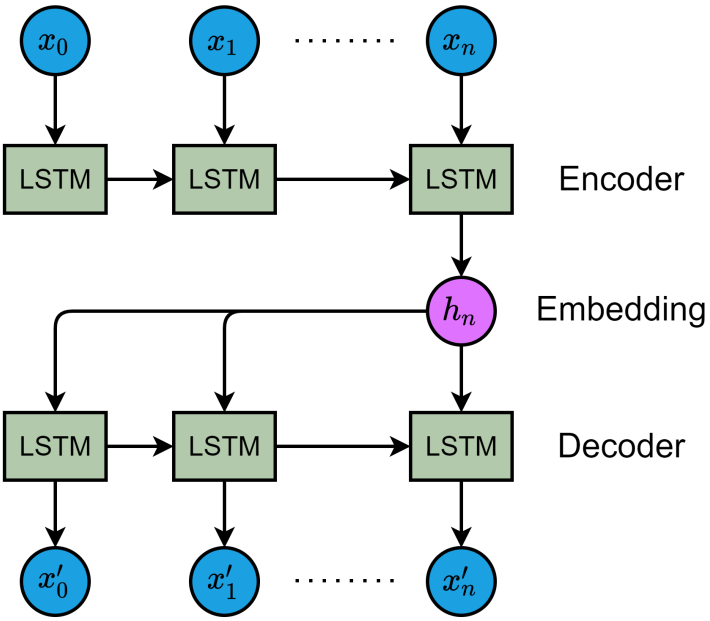

Figure 2.4: LSTM Autoencoder.

2.3 Clustering

K-means (MacQueen et al., 1967) is a vector quantization method that seeks to split n observations into k clusters, with each observation belonging to the cluster with the closest mean, which serves as the cluster's prototype. The method alternates between two steps, starting with an initial set of k means (clusters):

- Assign each observation to the cluster with the closest mean.
- Recalculate the means (centroids) for each cluster's observations.

When the assignments no longer change, the algorithm converges. There are multiple possible strategies for initializing means. In this thesis, we consider *kmeans++* (Arthur et al., 2007) as an initialization method. The example of clustered two-dimensional points is illustrated in Figure 2.5.

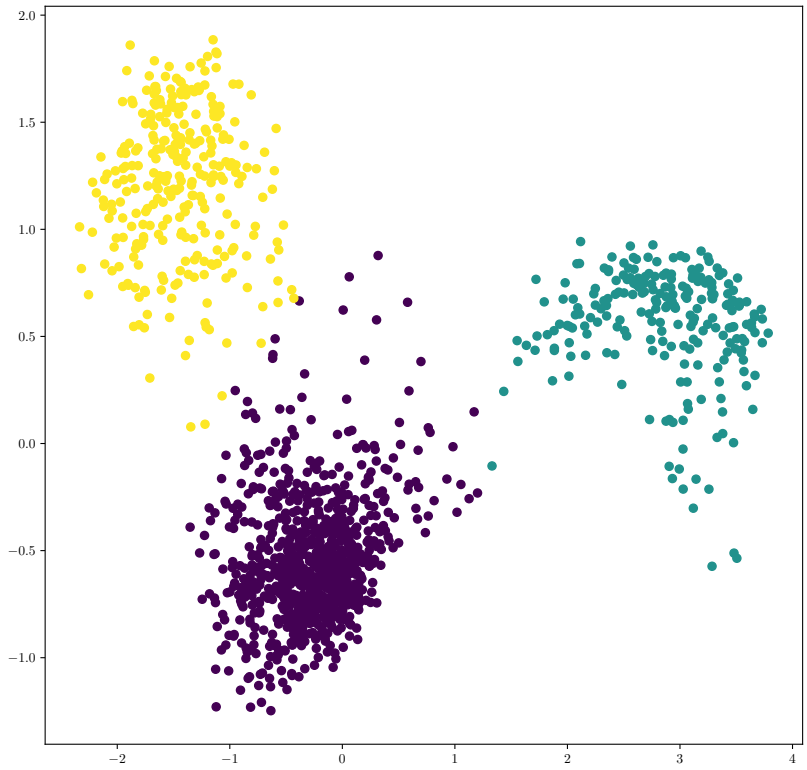

Figure 2.5: K-means clustering example.

2.4 Feature importance

Feature importance refers to methods for calculating a score for each of a model's input features; the scores describe the "importance" of each feature. A higher score indicates a more significant impact on the model used to predict a particular variable. Models and algorithms used for computing feature importance scores are described in Section 4.3.3.

Chapter 3

Related work

Various approaches have been used to analyze players' performance. Neural networks (Do et al., 2021) were able to predict the game outcome with 75.1% accuracy by aggregating random players' statistics across recent ranked matches. Their dataset includes champion mastery points, player-champion win rate, the total number of games played on the champion, and the number of recent games played on the champion. Even though they showed promising results, their methods do not consider the games' order. (Do et al., 2021) use information about all competitors. At the same time, we propose filtering out the features of all other participants except the analyzed one.

A noticeably different approach has been used by "considering the characteristics of gameplay over time" (S.-K. Lee et al., 2020). Their model tries to predict the game outcome based on changes over time and dynamic features. As expected, the longer the match, the more accurately the model can predict the outcome, with more than 96% accuracy and precision for 30+ minutes long matches (approximate duration of a match in LoL is 26-30 minutes[2]).

The drafting consists of picking and banning phases, essential parts of a match. Game balance and design are always imperfect, and the match might be won before it even starts by exploiting the game's current state. (Costa et al., 2021) clearly shows that picked champions concerning players' statistics have a massive impact on the game outcome. In contrast, information about picked and banned champions alone is not enough for reasoning. It is fair to say that their dataset includes games of professional players only, so it is inappropriate to discuss such results for regular players.

[2] `https://www.leagueofgraphs.com/rankings/game-durations`

(Jiang et al., 2021) developed a method to create different types of embeddings for predicting players' performance and game outcomes. However, Riot Games API (Riot Games, 2022) has evolved since then, and much new information can be obtained now using the most recent version of the API. Thus, our models are trained on data obtained from the most recent matches. Also, (Jiang et al., 2021) takes into account different game versions and champion types. In contrast, our approach does not use this information since we attempt to cluster players based on their statistics across different patch versions, champions, and roles.

Chapter 4

Methodology

This chapter describes how the dataset is created, what data is analyzed, and what models and algorithms are used.

4.1 Dataset creation

LoL uses the Elo rating system, which divides the player base into nine different tiers plus an unranked one. This thesis analyzes only the highest ones because they have a more consistent playstyle, deep understanding of game mechanics, and solid skills. Riot Games provides a public API (Riot Games, 2022) that we used to query random players from the three highest tiers on the EUW server. Then we pulled their recent ranked matches, where the JSON dictionary represents each match. Because of an evolution of the API, not all matches had the data we needed, and therefore they have been filtered out. In addition, one of our models cannot handle variable sequence length. Hence the number of matches per player is limited to 100. A total of 141 300 matches' data (1413 players) were collected for the period between 2022-02-02 and 2022-04-14.

As a consequence of how API data is updated internally, pulled data contains duplicates that were removed. Many insignificant match data were filtered out, such as strings (map id, queue id, names of players), runes and summoner spells (modifications that the player selects at the start), items, a champion, and a role. As we stated previously, our solution focuses only on one player and their history, and it does not involve other participants' data. The total number of remaining features is 202. The dataset overview is shown in Table 4.1

	Type	Size
Raw data	Json dictionary	1,413 players (141,300 matches)
Dataset	CSV	1,396 players (139,600 matches)

Table 4.1: The dataset overview.

We split the dataset into training and test sets with the most common ratio of 80:20. After that, 20% of the training set is separated into the validation set. These sets are used in a usual way:

- The training set is used to fit a model.
- Metrics calculated on the validation set provide an unbiased model evaluation. The validation set is used for hyperparameters tuning and over-fitting detection.
- The test set finally provides an unbiased evaluation of the tuned model.

Missing values are filled using three techniques: zero imputation, mean imputation, and median imputation. Lastly, all features were scaled using z-score normalization.

4.2 Feature understanding

Our dataset consists of 202 features. This section provides a short description for the vast majority of them to understand better the data we are working with. Features are grouped and explained in the following list:

- Different types of kills, including those under the enemy turret, consecutive ones (killing sprees, multi-kills), kills with the help from the epic monster, kills in the jungle, kills after flashing, multi-kills using just one spell, kills after being camouflaged, the first kill in the game.
- Different types of assists, including knocking an enemy into a player's team, assist streaks, and the first blood assist.
- Experience and gold earned during a match, number of items (including wards) purchased, and gold spent. Champion's level and the level advantage over the opponents. A killed number of minions and jungle monsters.

- Different types of damage (physical, magic, true, critical) dealt to enemies, neutral minions, epic monsters, and structures. Different types of damage taken from all sources. Damage mitigated using shields and unique passives.
- Healing and shielding teammates, saving them from death.
- Vision score, different wards (stealth, control, detector, sight) purchased and placed. Wards' area and time coverage. Enemy wards killed and deactivated.
- Number of times each ability and spell was used.
- Number of epic monsters (dragons, heralds, barons) kills, takedowns and steals. Structures (turrets, inhibitors, nexus) kills and takedowns. Structures killed by enemies.

Features not listed above are too tied to this game and require a deep knowledge of game mechanics. Moreover, some of them are not documented in the API. It is worth noting that features are not independent, and some of them are combinations of the other ones.

4.3 Modeling

This section briefly describes used models for creating embeddings, clustering, and anomaly (outlier) detection. We also provide information about their architectures, optimizers, and used loss functions.

4.3.1 Embeddings

Two different deep learning models have been tried to encode players' match histories into embeddings: Multilayer Autoencoder and Stacked LSTM Autoencoder. The Multilayer Autoencoder is used as a baseline model because of its simplicity and ability to encode any input data.

4.3.1.1 Multilayer Autoencoder

A Multilayer Autoencoder is a deeper version of a Vanilla Autoencoder. The encoder and the decoder consist of fully-connected layers with activation functions. Because this type of network does not directly work with sequences, we flattened them before feeding. The input is represented as a flattened sequence of matches' features; 202 features and 100 matches

= 20200 is the size of the input layer. Using fully-connected layers only, this model has many parameters and is expensive to train compared to other used models. Backpropagation with an Adam optimizer (Kingma et al., 2014) and an MSE loss function is the used training algorithm. The architecture is illustrated in Figure 4.1

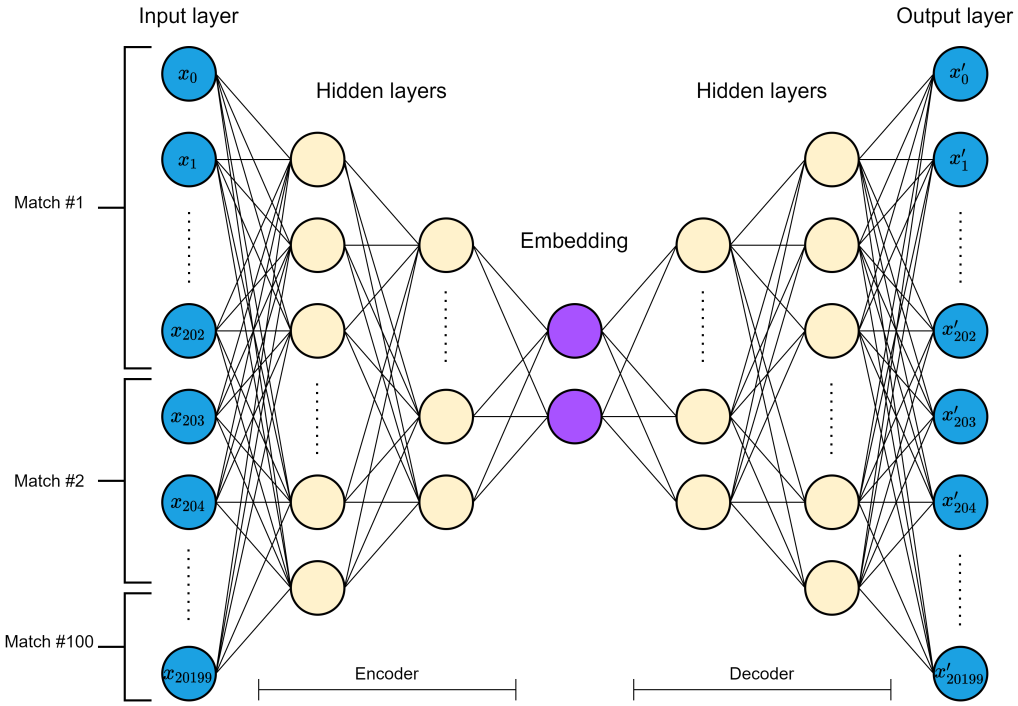

Figure 4.1: Multilayer Autoencoder.

4.3.1.2 Stacked LSTM Autoencoder

A Stacked LSTM Autoencoder is an LSTM autoencoder with multiple LSTM layers in the encoder and the decoder. All layers in the encoder and the decoder can have different output dimensions but the last ones. A stacked LSTM can handle longer sequences than a Vanilla LSTM network. We used an Adam optimizer and an MSE loss to train this model. The architecture is shown in Figure 4.2

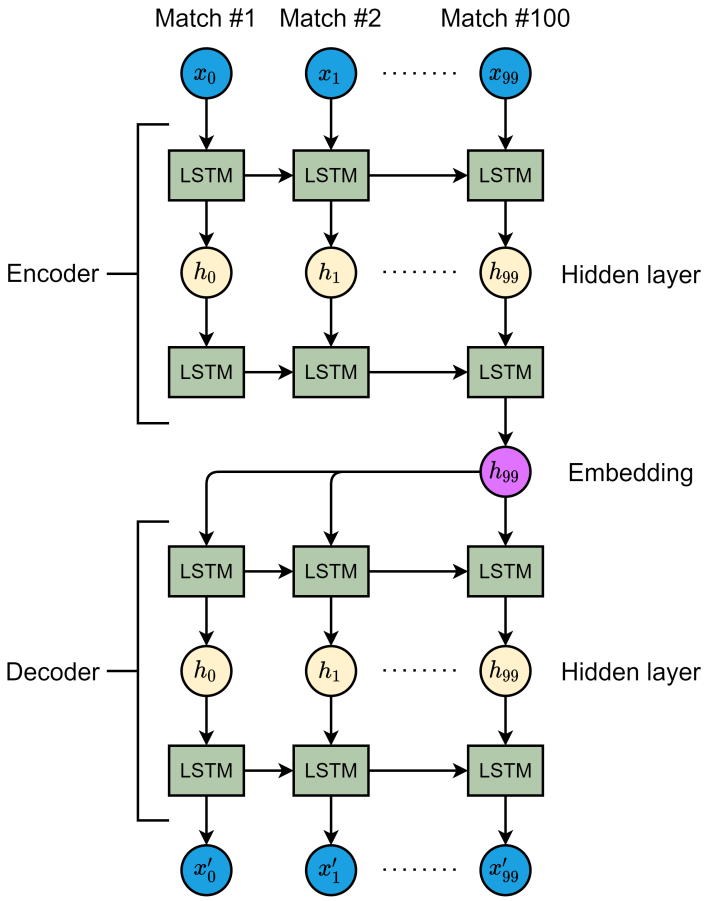

Figure 4.2: Stacked LSTM Autoencoder.

4.3.2 Unsupervised Anomaly Detection model

This model is proposed by (Souček, 2020) and was used for log anomaly detection. Although the model is called Unsupervised Anomaly Detection, the basic idea is to predict the following game statistics given previous ones. This model also utilizes LSTM layers but takes the output from all steps and passes it to linear layers with activation functions. Suppose such a model can predict the following game statistics with high accuracy. In that case, it can be used for outlier detection or, in other words, for finding suspicious matches in a player's history. The threshold is used to determine if the subsequent match statistics are different enough to conclude that there is an anomaly. The architecture of the model is visualized in Figure 4.3.

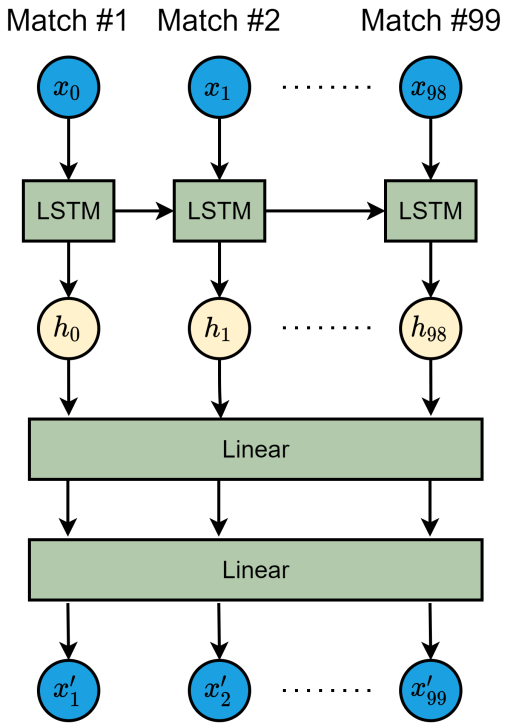

Figure 4.3: Unsupervised Anomaly Detection model.

4.3.3 Clustering and feature importance

Clustering was done on the generated embeddings from the previous step using the K-means algorithm. In order to determine the optimal number of clusters, we used the elbow method (Marutho et al., 2018). We also used Principal component analysis (PCA) (Pearson, 1901) for dimensionality reduction, where each embedding is projected onto the first two components, such that we can provide visualization of resulting clusters.

To analyze and investigate the clusters, we averaged the match statistics of the players and then fitted a logistic regression (Cox, 1958) with the clusters as the labels. We used logistic regression weights as the feature importance scores.

Chapter 5

Experiments

In this chapter, we consider our experiments' setup and the results of different stages of our pipeline.

5.1 Setup

The programming language was `Python 3.9` (Van Rossum et al., 2009), frameworks for preparing dataset were `pandas`[3], `numpy`[4] and `scikit-learn`[5]. Clustering algorithms are provided by `scikit-learn` library. We selected `pytorch` (Paszke et al., 2019) and its wrapper `pytorch lightning`[6] as the machine learning libraries. We tracked the experiments using `Weights & Biases`[7] cloud service. We also used `Jupyter Lab`[8] as an interactive python environment and `matplotlib`[9] as a visualization tool.

We ran our experiments on the RCI computing cluster[10]. The hardware used for training models had the following specifications: Intel Xeon Scalable Gold 6150, 32GB RAM, Tesla V100 32GB with NVLink. The seed for random number generation is set to 42 for all of our experiments.

[3] `https://pandas.pydata.org/`
[4] `https://numpy.org/`
[5] `https://scikit-learn.org/`
[6] `https://pytorchlightning.ai/`
[7] `https://wandb.ai/`
[8] `https://jupyter.org/`
[9] `https://matplotlib.org/`
[10] `http://rci.cvut.cz/`

5.2 Results

5.2.1 Embeddings

In order to cluster and extract feature importance information, we first need to create embeddings for each player. So we started our experiments by training different types of Autoencoders and selecting the best architectures and parameters for them. The standard parameters for all models are embedding size and imputation method. The primary metric for the models was MSE loss on the validation split.

The first model we have tried was Multilayer Autoencoder. We decided to use three architectures that vary in depth and size of the final model. The total number of configurations is 36. The complete list of results is shown in Table 5.1. As the imputation method might significantly impact the model's performance, the best model using zero imputation achieved a validation loss of 1.108. In contrast, the best model across all imputation methods has reached a validation loss of 0.861. It is also worth noting that the best model is the widest one as it has only one linear layer with a size of 5000 in the encoder and the decoder. However, the embedding size is 50, which is not the largest. With certain confidence, we can claim that this embedding size is enough to encode players' last 100 matches in the case of Multilayer Autoencoder. The is almost no difference between the median and mean imputation; both provide the same results with a slight mean method advantage.

The second model used for creating embeddings was Stacked LSTM Autoencoder. Our Stacked LSTM Autoencoder had 2 LSTM layers in the encoder and the decoder with varying hidden sizes. The total number of configurations is 48. The complete list of results is shown in Table 5.2. The best model (mean imputation, embedding size = 25, hidden size = 500) is comparable with the best Multilayer Autoencoder, with a validation loss of 0.85. Nevertheless, the difference in the number of parameters is huge, three million in Stacked LSTM Autoencoder against 202 million in Multilayer Autoencoder. A recurrent LSTM network with fewer parameters can capture important game statistics and reproduce them from the hidden state on the same level as a Multilayer Autoencoder.

Imputation method	Embedding size	Hidden layers' sizes	Train loss	Validation loss
zero	25	[1000,1000,500]	0.661	1.153
		[2500,500]	0.047	1.287
		[5000]	0.077	1.337
	50	[1000,1000,500]	0.638	1.172
		[2500,500]	0.053	1.265
		[5000]	0.062	1.236
	100	[1000,1000,500]	0.670	1.150
		[2500,500]	0.054	1.254
		[5000]	0.056	1.203
	200	[1000,1000,500]	0.660	1.162
		[2500,500]	0.059	1.265
		[5000]	0.081	1.108
median	25	[1000,1000,500]	0.485	0.931
		[2500,500]	0.220	0.927
		[5000]	0.117	0.894
	50	[1000,1000,500]	0.491	0.924
		[2500,500]	0.078	0.924
		[5000]	0.318	0.871
	100	[1000,1000,500]	0.548	0.916
		[2500,500]	0.075	0.933
		[5000]	0.135	1.337
	200	[1000,1000,500]	0.576	0.905
		[2500,500]	0.047	0.934
		[5000]	0.760	0.886
mean	25	[1000,1000,500]	0.503	0.917
		[2500,500]	0.218	0.925
		[5000]	0.184	0.870
	50	[1000,1000,500]	0.535	0.920
		[2500,500]	0.107	0.932
		[5000]	0.095	0.861
	100	[1000,1000,500]	0.569	0.904
		[2500,500]	0.087	0.934
		[5000]	0.222	0.877
	200	[1000,1000,500]	0.544	0.908
		[2500,500]	0.121	0.939
		[5000]	0.894	0.971

Table 5.1: Multilayer Autoencoder's performance.

Imputation method	Embedding size	Hidden size	Train loss	Validation loss
zero	25	100	1.461	1.536
		500	1.479	1.551
		1000	1.583	1.670
		2000	1.583	1.659
	50	100	1.484	1.554
		500	1.458	1.537
		1000	1.462	1.540
		2000	1.552	1.622
	100	100	1.467	1.543
		500	1.454	1.535
		1000	1.453	1.539
		2000	1.517	1.587
	200	100	1.473	1.546
		500	1.449	1.535
		1000	1.441	1.536
		2000	1.476	1.550
median	25	100	0.762	0.860
		500	0.749	0.856
		1000	0.771	0.854
		2000	0.827	0.902
	50	100	0.760	0.854
		500	0.744	0.856
		1000	0.732	0.863
		2000	0.777	0.862
	100	100	0.762	0.853
		500	0.749	0.855
		1000	0.738	0.858
		2000	0.717	0.869
	200	100	0.758	0.853
		500	0.745	0.856
		1000	0.747	0.855
		2000	0.743	0.865
mean	25	100	0.760	0.855
		500	0.764	0.850
		1000	0.768	0.852
		2000	0.798	0.871
	50	100	0.759	0.855
		500	0.749	0.854
		1000	0.748	0.853
		2000	0.772	0.855
	100	100	0.757	0.852
		500	0.742	0.853
		1000	0.748	0.857
		2000	0.760	0.854
	200	100	0.758	0.852
		500	0.748	0.853
		1000	0.737	0.859
		2000	0.784	0.858

Table 5.2: Stacked LSTM Autoencoder's performance.

5.2.2 Unsupervised Anomaly Detection model

Because zero imputation showed poor results, we have decided to exclude it from unsupervised anomaly detection experiments. As shown in Table 5.3, the model is not capable of predicting the next match statistics. The results were expected because there is much randomness in matchmaking algorithms and other players' behavior. This architecture can probably be used for predicting a subset of match statistics. However, this is beyond the scope of this thesis.

Imputation method	LSTM layers	Linear layers	Linear size	Train loss	Validation loss
median	4	4	100	0.592	0.993
			300	0.556	0.998
			500	0.564	1.015
		8	100	0.614	0.993
			300	0.623	0.996
			500	0.628	1.004
	8	4	100	0.710	0.927
			300	0.732	0.900
			500	0.647	0.998
		8	100	0.847	0.923
			300	0.895	0.973
			500	0.895	0.973
	32	4	100	0.894	0.972
			300	0.894	0.972
			500	0.894	0.972
		8	100	0.895	0.973
			300	0.895	0.973
			500	0.895	0.973
mean	4	4	100	0.580	0.998
			300	0.554	1.004
			500	0.574	1.001
		8	100	0.629	0.997
			300	0.632	0.998
			500	0.638	0.976
	8	4	100	0.696	0.939
			300	0.657	0.985
			500	0.657	0.984
		8	100	0.781	0.863
			300	0.893	0.970
			500	0.893	0.970
	32	4	100	0.892	0.970
			300	0.892	0.970
			500	0.893	0.970
		8	100	0.893	0.970
			300	0.893	0.970
			500	0.893	0.970

Table 5.3: Unsupervised Anomaly Detection model performance.

5.2.3 Clustering and feature importance

5.2.3.1 Multilayer Autoencoder

The optimal number of clusters according to the elbow method is five. The visualization of the first two PCA components is shown in Figure 5.1. The green cluster may represent outliers since it spreads over a large area, whereas the remaining clusters overlap. We picked twenty feature importance scores for each cluster, ten with the maximum positive values and ten with the maximum negative values. Feature importance scores for three clusters are presented in Figure 5.2. The remaining scores are in Figure A.1

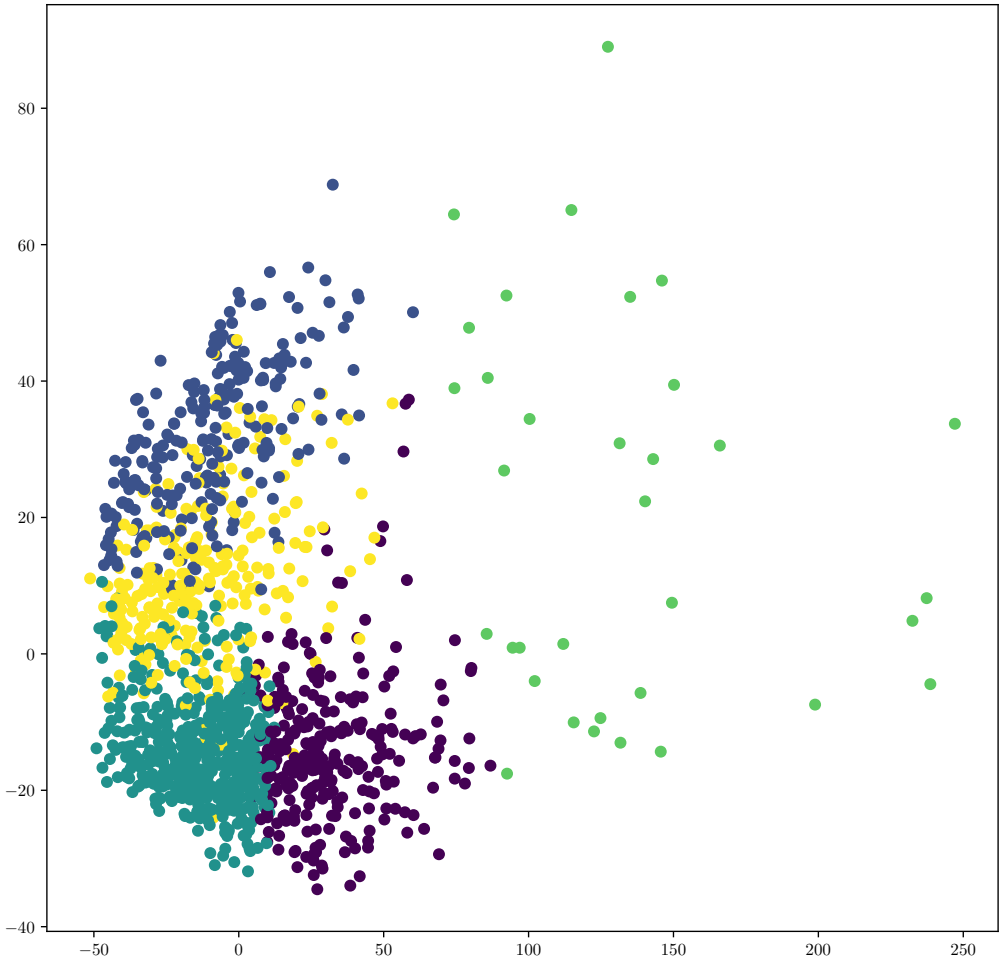

Figure 5.1: Multilayer Autoencoder clusters.

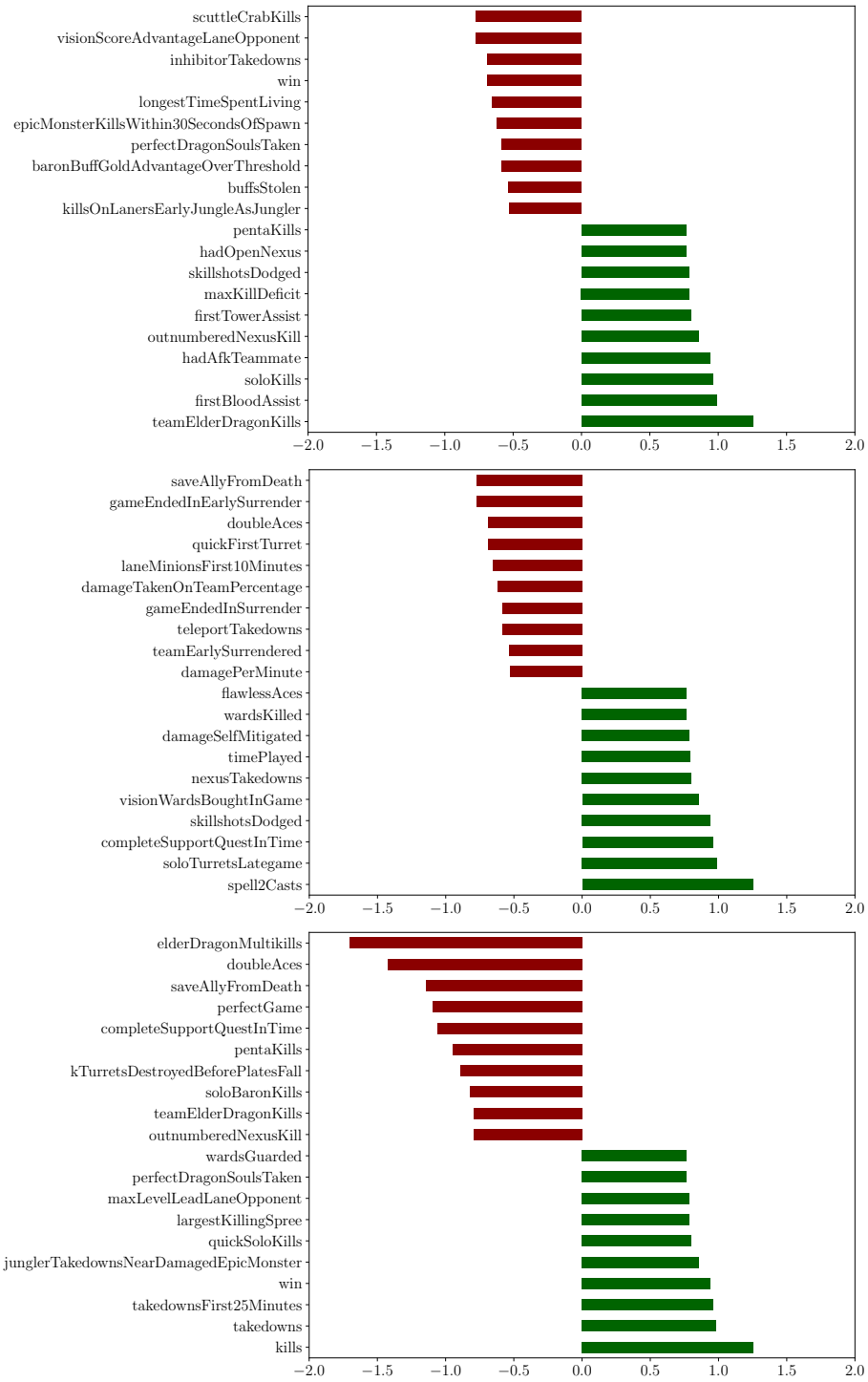

Figure 5.2: Multilayer Autoencoder feature importance scores (1-3).

5.2.3.2 Stacked LSTM Autoencoder

According to the elbow approach, the optimal number of clusters is three. Figure 5.3 illustrates the first two PCA components of the produced embeddings. Stacked LSTM Autoencoder shows better performance than Multilayer Autoencoder as clusters are more separated and do not overlap. The feature importance scores are visualized in Figure 5.4.

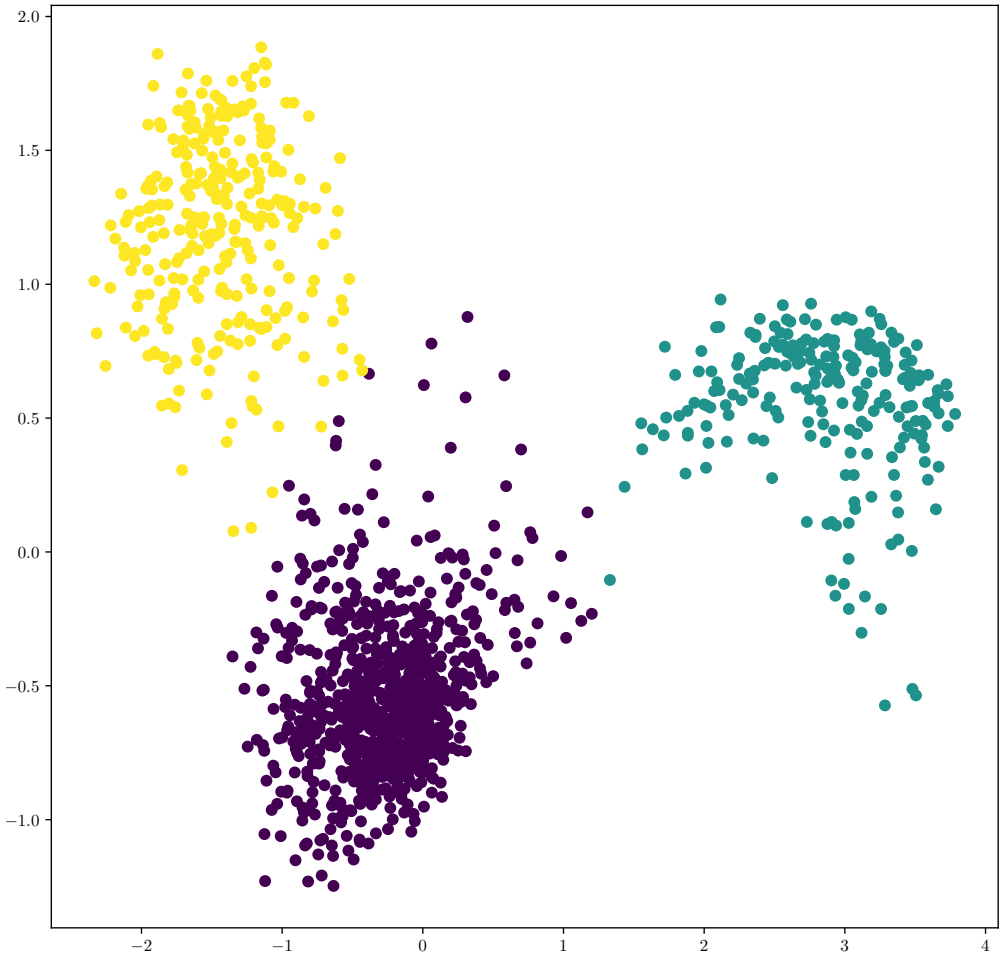

Figure 5.3: Stacked LSTM Autoencoder clusters.

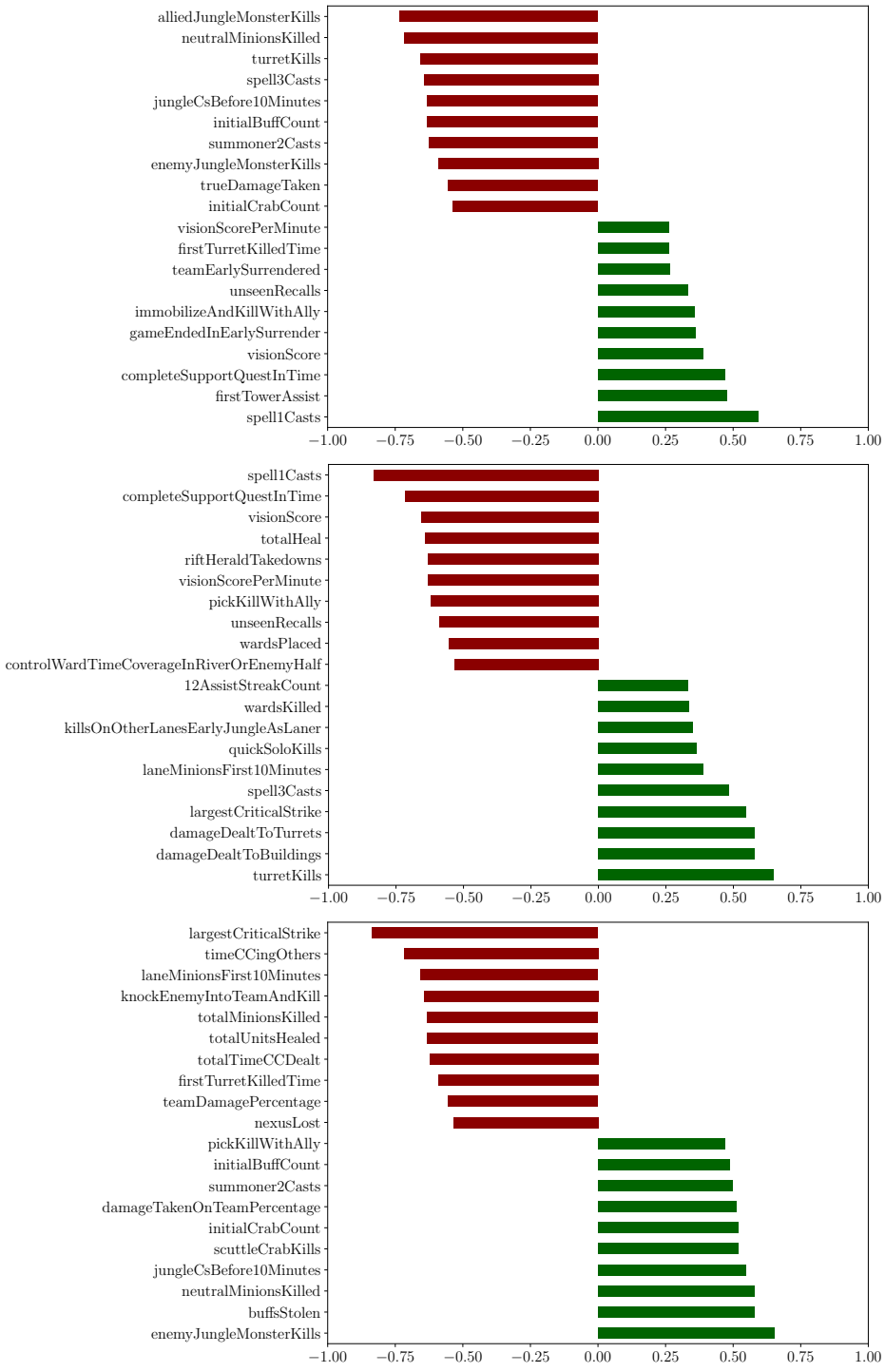

Figure 5.4: Stacked LSTM Autoencoder feature importance scores.

5.3 Discussion

Even though autoencoders' performance can be evaluated by validation loss, the quality of the formed clusters cannot be automatically determined. Therefore we analyzed them manually.

Multilayer Autoencoder produced five clusters. The following is a list of their respective interpretations:

- Solokillers and first blood lovers, whose teammates often abandon the game, which might imply that they misbehave in the chat and force other players to leave the game. They have an aggressive playstyle with many fast consecutive kills and fights around objectives. Their win rate is the lowest across all clusters; they frequently die and rarely place wards.
- Turrets destroyers, who tend to have long games with a high amount of placed and killed wards. They are weak in the early game, do not like grouping with their team, and dodge many opponents' skillshots. They manage to mitigate much damage using shields and healing.
- Players with a high win rate and many kills, especially in the early game. They always have a level advantage on their lane opponent, longest killing sprees without dying, and they tend to kill many dragons. They do not like playing supports and do not have fast consecutive kills.
- Epic monsters slayers, who save allies and provide crowd control for their team. They prioritize efficient movement around the map. They are bad at skill dodging, and they do not have a lead during the laning phase.
- Farming junglers with the highest damage in their team. They successfully attack laners, kill dragons, and manage to lead in experience and gold. They do not tend to skirmish in the early game and have insignificant damage to turrets.

Stacked LSTM Autoencoder produced three clusters. The following is a list of their interpretations:

- Supports with a lot of placed wards and first spell casts. They tend to assist with the first turret kill and often recall to the base. They are the source of crowd control for their team. They do not farm minions and jungle monsters.

- Turret killers with high damage to buildings. They rely on their third spell, have the largest critical strikes, and love quick solo kills. They also tend to move between lanes looking for enemy champions and wards.
- High damage junglers with many jungle monsters stolen from opponents. They try to outfarm the enemy jungler, counter gank them, and tank most of the damage.

The interpretations given above are meant for players with respected game knowledge. They could potentially provide insights and improve players' performance.

Chapter 6

Conclusion

In this thesis, we introduced the problem of player modeling in computer games. We chose to conduct experiments on the popular MOBA game - League of Legends. The main concepts employed in our experiments are described in Chapter 2. The dataset was created by pulling game matches from Riot Games API, and collected data was preprocessed using several techniques. We provided an overview of the most crucial features of the dataset in Section 4.3.3. Two different models were employed to create embeddings. Generated embeddings were used to cluster the individual players, and we discussed assigned clusters with respect to the features' importance scores and playing style in general.

6.1 Limitations

- Riot Games API has a rate limit, so the dataset is restricted in size.
- We used only 100 most recent ranked games for each player, while our models can handle much longer sequences.
- Although we filtered out important features, such as runes, summoner spells, champions, and roles, models do not thoroughly understand these features, i.e., there are 159 champions in LoL with unique abilities, attributes, and interactions.

6.2 Future work

- Alternative models could be used to create embeddings, e.g., Transformers, or Convolutional Autoencoders.
- The produced embeddings can be used to predict match outcomes, track playstyle changes and detect suspicious behavior.
- It should be noted that the results of this thesis could potentially be used to create an online service for players and esports coaches.

Bibliography

ARTHUR, David; VASSILVITSKII, Sergei, 2007. K-Means++: The Advantages of Careful Seeding. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, pp. 1027–1035. SODA '07. ISBN 9780898716245.

COSTA, Lincoln Magalhães; MANTOVANI, Rafael Gomes; MONTEIRO SOUZA, Francisco Carlos; XEXÉO, Geraldo, 2021. Feature Analysis to League of Legends Victory Prediction on the Picks and Bans Phase. In: *2021 IEEE Conference on Games (CoG)*, pp. 01–05. ISSN 2325-4289. Available from DOI: 10.1109/CoG52621.2021.9619019.

COX, David R, 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*. Vol. 20, no. 2, pp. 215–232.

DO, Tiffany D; WANG, Seong Ioi; YU, Dylan S; MCMILLIAN, Matthew G; MCMAHAN, Ryan P, 2021. Using machine learning to predict game outcomes based on player-champion experience in league of Legends. Available from arXiv: 2108.02799 [cs.LG].

GREFF, Klaus; SRIVASTAVA, Rupesh K.; KOUTNIK, Jan; STEUNEBRINK, Bas R.; SCHMIDHUBER, Jurgen, 2017. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*. Vol. 28, no. 10, pp. 2222–2232. Available from DOI: 10.1109/tnnls.2016.2582924.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen, 1997. Long Short-term Memory. *Neural computation*. Vol. 9, pp. 1735–80. Available from DOI: 10.1162/neco.1997.9.8.1735.

JIANG, Julie; LERMAN, Kristina; FERRARA, Emilio, 2021. Individualized context-aware tensor factorization for online games predictions. Available from arXiv: 2102.11352 [cs.AI].

KINGMA, Diederik P.; BA, Jimmy, 2014. *Adam: A Method for Stochastic Optimization*. arXiv. Available from DOI: 10.48550/ARXIV.1412.6980.

LEE, Hojoon; HWANG, Dongyoon; KIM, Hyunseung; LEE, Byungkun; CHOO, Jaegul, 2022. DraftRec: Personalized Draft Recommendation for Winning in Multi-Player Online Battle Arena Games. In: *Proceedings of the ACM Web Conference 2022*. ACM. Available from DOI: 10.1145/3485447.3512278.

LEE, Sang-Kwang; HONG, Seung-Jin; YANG, Seong-Il, 2020. Predicting Game Outcome in Multiplayer Online Battle Arena Games. In: *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1261–1263. ISSN 2162-1233. Available from DOI: 10.1109/ICTC49870.2020.9289254.

MACQUEEN, James et al., 1967. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Oakland, CA, USA. Vol. 1, pp. 281–297. No. 14.

MARUTHO, Dhendra; HENDRA HANDAKA, Sunarna; WIJAYA, Ekaprana; MULJONO, 2018. The Determination of Cluster Number at k-Mean Using Elbow Method and Purity Evaluation on Headline News. In: *2018 International Seminar on Application for Technology of Information and Communication*, pp. 533–538. Available from DOI: 10.1109/ISEMANTIC.2018.8549751.

OLAH, Christopher, 2015. *Understanding LSTM Networks*. Available also from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KÖPF, Andreas; YANG, Edward; DEVITO, Zach; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith, 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv. Available from DOI: 10.48550/ARXIV.1912.01703.

PEARSON, Karl, 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. Vol. 2, no. 11, pp. 559–572. Available from DOI: 10.1080/14786440109462720.

RIOT GAMES, 2022. *Riot Developer Portal* [online]. [visited on 2022-04-24]. Available from: https://developer.riotgames.com/apis.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J., 1986. Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, pp. 318–362. ISBN 026268053X.

RUMELHART, David E.; MCCLELLAND, James L., 1987. Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pp. 318–362.

SOUČEK, Marek, 2020. *Log Anomaly Detection*. MA thesis. Czech Technical University in Prague. Computing and Information Centre.

VAN ROSSUM, Guido; DRAKE, Fred L., 2009. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. ISBN 1441412697.

WENG, Lilian, 2018. From Autoencoder to Beta-VAE. *lilianweng.github.io*. Available also from: https://lilianweng.github.io/posts/2018-08-12-vae/.

Appendix A

Figures

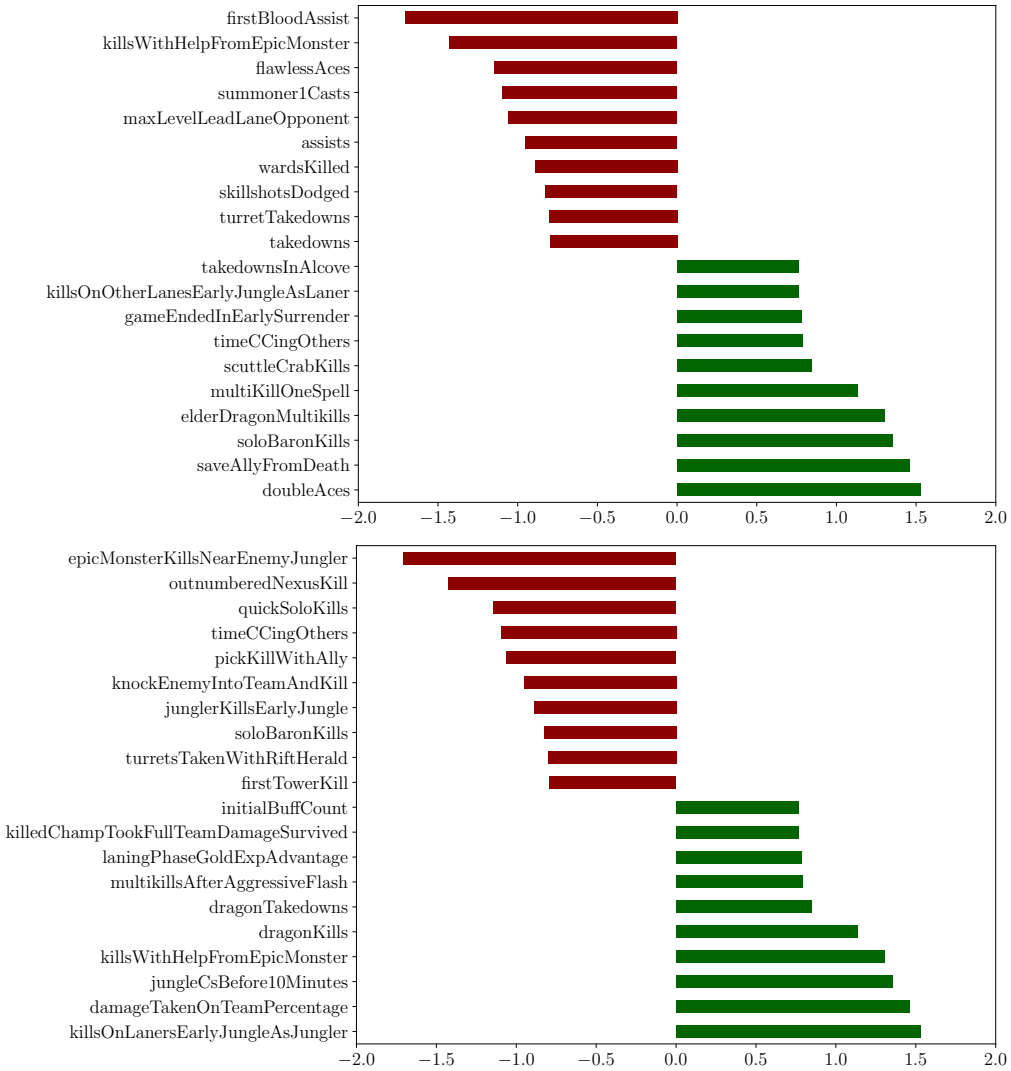

Figure A.1: Multilayer Autoencoder feature importance scores (4-5).

Appendix B

Acronyms

MOBA Multiplayer online battle arena

LoL League of Legends

RNN Recurrent neural network

LSTM Long short-term memory

PCA Principal component analysis

Appendix C

Source code structure

```
readme.md ..... the file with source code description
thesis.pdf ..... the thesis text in PDF format
thesis ..... the directory of LaTeX source codes of the thesis
implementation ..... the directory with implementation sources
```