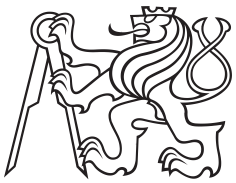
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Predicting Counter-Strike Game Outcomes with Machine Learning

Ondřej Švec

Supervisor: Ing. Gustav Šír, Ph. D.
Field of study: Cybernetics and Robotics
January 2022

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Švec Ondřej** Personal ID number: **483800**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Predicting Counter-Strike Game Outcomes with Machine Learning

Bachelor's thesis title in Czech:

Predikce výsledků ve hře Counter-Strike s pomocí strojového učení

Guidelines:

The e-sports industry experiences an unprecedented growth in popularity and impact, however, predictive analytics studies in the domain are still considerably rare. Moreover, in contrast to classic sports matches, there are also rich data records collected from the virutal game enviroment, allowing for a range of machine learning
methods to be tested. The subject of this thesis is such an exploratory machine learning study into the most popular e-sport - CS:GO.
1) Review the state of predictive analytics in CS:GO.
2) Collect statistically significant amount of match data.
3) Perform exploratory data analysis and preprocessing.
4) Experiment with a range of sample representations and models.
5) Analyse and compare your results against selected baselines.

Bibliography / sources:

[1] Björklund, Arvid, et al. Predicting the outcome of CS: GO games using machine learning. BS thesis. Chalmers University of Technology, 2018.
[2] Makarov, Ilya, et al. "Predicting winning team and probabilistic ratings in "Dota 2" and "Counter-Strike: Global Offensive" video games." International Conference on Analysis of Images, Social Networks and Texts. Springer, Cham, 2017.
[3] Minka, Tom, Ryan Cleven, and Yordan Zaykov. "Trueskill 2: An improved bayesian skill rating system." Tech. Rep. (2018).
[4] Zhou, Jie, et al. "Graph neural networks: A review of methods and applications." AI Open 1 (2020): 57-81.

Name and workplace of bachelor's thesis supervisor:

Ing. Gustav Šír, Ph.D., Department of Computer Science, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **20.05.2021** Deadline for bachelor thesis submission: **04.01.2022**

Assignment valid until: **19.02.2023**

Ing. Gustav Šír, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to thank my supervisor Ing. Gustav Šír, Ph. D. greatly for leading me throughout the year.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 4. January 2022

Abstract

The surge of the esports industry, alongside the rise in popularity of machine learning, provides unique opportunities for improving currently used methods of match outcome predictions.

This thesis aims to give a new perspective to predicting outcomes of Counter-Strike matches. In order to achieve that, first, high volume dataset consisting of over 40,000 matches was obtained and analyzed. Three sample representations, two types of data preprocessing, six machine learning models and a model utilizing Elo rating were proposed, cross-validated and compared to a selected baseline. Model utilizing Elo rating proved to be consistently the best one, with test set accuracy of 64 %, closely followed by Random Forest model with test set accuracy of 63 %.

Keywords: machine learning, neural networks, Counter-Strike, esports, predicting outcomes

Supervisor: Ing. Gustav Šír, Ph. D.

Abstrakt

Vzestup odvětví esportů spolu s nárůstem popularity strojového učení poskytuje jedinečné příležitosti pro zlepšení aktuálně používaných metod předpovědí výsledků zápasů.

Tato práce si klade za cíl poskytnout nový pohled na predikci výsledků zápasů Counter-Strike. Aby toho bylo dosaženo, byl nejprve získán a analyzován velký objem dat skládající se z více než 40 000 zápasů. Byly navrženy tři reprezentace dat, dva typy preprocessingu dat, šest modelů strojového učení a model využívající hodnocení Elo, které byly křížově ověřeny a porovnány s vybraným výchozím stavem. Model využívající hodnocení Elo se prokázal být nejlepším, s přesností na testovací sadě 64 %, těsně následovaný modelem Random Forest s přesností na testovací sadě 63 %.

Klíčová slova: strojové učení, neuronové sítě, Counter-Strike, esports, předpověď výsledků

Překlad názvu: Predikce výsledků ve hře Counter-Strike s pomocí strojového učení

Contents

Figures

Tables

Chapter 1

Introduction

Sports have been around us for as long as we can remember, with the documented history going back thousands of years. What may have started as one of the ways to prepare for an incoming war or perhaps a hunt, became an important part of today's cultural society. Over the years, sports have developed from throwing spears and rocks in a numerous different ways for people to compare their skills [1]. Some of these sparking up a discussion about what exactly sport is, what it embodies and what it does not. For some, sporting is an activity involving physical exertion and competition, for others the latter suffices [2].

With the invention of the computer and the internet, possibly the latest addition to sports arose - esports. Often talked in the context of the fastest growing sports industry (placed as the best sport rated by potential to grow revenues globally by PwC in 2018 [3], 2019 [4] and 2020 [5]), esports' worldwide market revenues estimates rose from 130 million U.S. dollars to nearly 1.1 billion U.S. dollars, making the predictions for 2024 reach over 1.6 billion U.S. dollars [6]. [1] The grow is has caught interest of traditional sports' organizations looking to diverge. In the last couple of years, sports organisations' level of engagement with esports grew rapidly, evolving from scepticism to embracing [5].

Counter-Strike: Global Offensive (CS:GO for short) has been launched in 2012 and has since become the highest ranking game on Steam in terms of average players each month with approximately 549 thousand players [11].

[1] According to [7], the esports audience grew 9.5 % from 397.8 to 435.9 million of occasional viewers with 220.5 million of esports enthusiasts.

One of its biggest advantages is the simplicity of core game mechanics, unlike in many other cases of other esports titles, making it very easy to pick up both playing and watching the game. To this day, CS:GO holds a standard of how a competitive FPS game is supposed to be like.

Sports and gambling have always gone hand in hand. Ancient Greek artefacts indicate that there was betting already on the Olympic Games [8]. However, with the invention of online betting, sports gambling is currently experiencing a huge boom. What used to be a small market, now is a billion-dollar industry. Sports betting is recently more and more, in terms of revenue, being compared to traditional sports [9].

With computational power being more accessible every year, machine learning became very popular in nearly every specialization in our lives. Thanks to machine learning, areas, such as speak recognition or image recognition, were greatly accelerated. Every other cellphone has nowadays got a speech recognition function powered by a machine learning model, and each year a steps towards fully autonomous vehicles are taken. In relation to sports predicting and betting, other methods are still preferred to machine learning. However, some preliminary research has already been done in the area, with some positive results.

1.1 Problem and goals

Machine learning models showed promising results in recent studies, improving predictions of outcomes, both, in traditonal sports and esports. This thesis tries to come up with different sample representations and prediction models to predict outcomes of CS:GO matches.

In this study, we have chosen to tackle the problem directly as classification task, i.e. predicting the winner of the game as one of the competing teams. Proposed sample representations then consist of player representation consisting of player statistics calculated to round average, team representation consisting of roster's ability to win percentages of classified rounds, and representation combining the two representations.

The models selected to approach this task then consist of various neural networks, standard machine learning models and an Elo rating-based model.

Chapter 2

Background

In this chapter we provide the necessary background to the game of CS:GO and the various machine learning concepts that will be utilized later in the thesis.

2.1 CS:GO and betting

Intersection of esports and betting is not one to take lightly. Since its launch in July 2012 to July 2013, CS:GO had averaged up to 20 thousand players on a month-to-month basis [11]. In August 2013, the Arms Deal update came out, allowing for trading and, especially, betting of in-game items, which, in turn, skyrocketed the interest in CS:GO. A full year later, in August 2014, CS:GO averaged 133 thousand players. Adding one more year, the count rose up even higher, to 357 thousand average players [11]. CS:GO is not the only esports title to heavily rely on gambling. Nearly 59 % of the 2021 esports revenue streams, valued at 641 million $, was made of sponsorship [7]. Conventional industries are getting themselves involved in esports as sponsors from all around, though, one of the biggest supporting forces in the industry is the gambling industry [12]. Interestingly, not only the teams take up gambling sponsorships, the tournament organizers do, too [13].

2.2 State of predictive analytics in CS:GO

With machine learning's rise in popularity, attempts to implement it one way or another in connection with CS:GO have been made, however, not many in terms of predicting match outcomes. Some people try teaching bots play via unsupervised learning like human players, to either gain an advantage, or to substitute human players when missing [16].

In 2017, [17] tried predicting outcomes of matches by using features calculated from so-called after-plant situations. After-plant situations occur when the attacking side (T side) plants a bomb and the timer before explosion starts to countdown. Features are made of data from each second of the after-plant. The study harvests data from 162 map demos (demo contains all actions happening in the game) played from February 2016 to March 2017. It achieves a 62 % accuracy. In this thesis, rather than observing a small dataset and trying to predict based on very specific features, a huge dataset is obtained and more of a general approach, in terms of features chosen, is taken.

CS:GO allows to record statistics from any match played in the game. This means anyone can gather data by playing the game and saving demo files. And many third party sites also allow viewing detailed statistics of any player. In 2018, [18] analyzed 6,000 matches of 1,000 top players from matching site FaceIt.com[1]. The study's goal was to collect in-game statistics, group players into clusters and predict a winning team by the clusters present in each match's teams. It achieved accuracy of 65.11 %, which showed an improvement compared to predictions made solely by observing FaceIt's ranks. However, this method favors from the fact that FaceIt matching doesn't take clustering of players in consideration at all, meaning it's perfectly possible for a team to have players preferring the same role in the game. This seems to be hardly transferable to professional match predicting, as teams are not made up by an algorithm, but by players teaming up, themselves.

Interestingly enough, as opposed to the lack of studies linking CS:GO and machine learning predictions of outcomes, number of studies for MOBA[2] games exist.

[1] FaceIt is third-party service, where players queue up and get matched based on rank FaceIt calculates for each player based on their perfomance in previous matches.

[2] "Multiplayer Online Battle Arena", a popular genre of esports games

2.3 Rules of CS:GO

Counter-Strike is a team-based multiplayer first person shooter. Two teams of five players each compete in multiple rounds with the goal of winning enough rounds to win a map. Most matches are played in either a best-of-1, best-of-3, or best-of-5 (often shortened to just BO1, BO3 and BO5) setting, meaning, that winning majority of a given count of maps results in a win.

Each map takes place in one of seven in-game levels[3] called "Active Duty Map Pool". The order of maps in a match is set via a "vote-ban system" where teams first decide maps that they do not want to play by banning them and follow up by picking maps that are then played. Every level is substantially different from all the others.

A map starts by deciding which team plays which side first. The usual way to decide the starting sides is performing a single knife round in which both teams play with only knives as weapons. The winner then chooses the preferred starting side. Maps are in most cases played as a best-of-30. First fifteen rounds are played, then sides are changed and rest of the rounds is played out until one of the teams reaches a majority (sixteen) of rounds or the map comes to a tie (15-15 score)[4].

To win a round, a team has to fulfill one of a few conditions depending on the side the team is playing in the round. Attacking side (often called "Terrorist side" or "T side" for short) wins a round by successfully planting and exploding a bomb in one of designated areas in a level. Defending side's (often called "Counter-Terrorist side" or "CT side" for short) goal is to prevent the bomb from exploding by either defusing an already planted bomb or delaying the attacking team enough for the round to close as time runs out. Both sides also win the round by eliminating all players of the other team before reaching their goal.

Economy plays a crucial role in Counter-Strike. A round starts with both teams locked in "buy zones" for the duration of "buy time". Teams buy armor, utility and weapons in order to maximize the chances of winning the round. At the end of each round, both teams receive bonus money for next round. While the losing team receives a higher bonus, multiple member of the winning team usually survive the round, which results in keeping the bought equipment for the next round without having to rebuy it.

[3] In this thesis level and map is used interchangeably, as well, as map and match, depending on the context.

[4] In which case, an overtime is usually played.

2.4 Player ranking - Elo

Elo is a rating system developed by Arpad Elo and used since 1970. It was developed, and to this day used, for rating chess players [19]. Nowadays, Elo rating and its derivatives are used in countless of sports and competitions, though, usually tweaked.

At the beginning, all players start with Elo rating of 1500. After the match ends, resulting in one of the sides taking a win, Elo ratings are recalculated using following formula [20]:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}},$$
$$R'_A = R_A + K \cdot (S_A - E_A),$$

where R_A stands for rating of side A before the match, R_B for rating of side B before the match, R_A for rating of side A after the match, S_A the actual outcome of the match from side A's perspective (1 for a win, 0 for a loss, 0.5 for a tie), E_A is the probability of side A winning the match and K a chosen scaling factor. In this study, R_A, R_B are calculated as means of theirs respective players. To update the Elo values, difference $R'_A - R_A$ is added to each of the players' ratings.

In this thesis, Elo rating serves a role of non-machine learning model, for comparison purposes.

2.5 Machine Learning

In this section, background is laid out to the used machine learning tools. First sub-section explains non-neural network models used in this thesis, second describes elemental layers of neural network models, that have been used, with the last two subsections explaining loss function and optimizers.

2.5.1 Non-neural network models

With the ever-growing expansion of machine learning, countless of very advanced models exist. For the lack of studies in CS:GO's outcome prediction

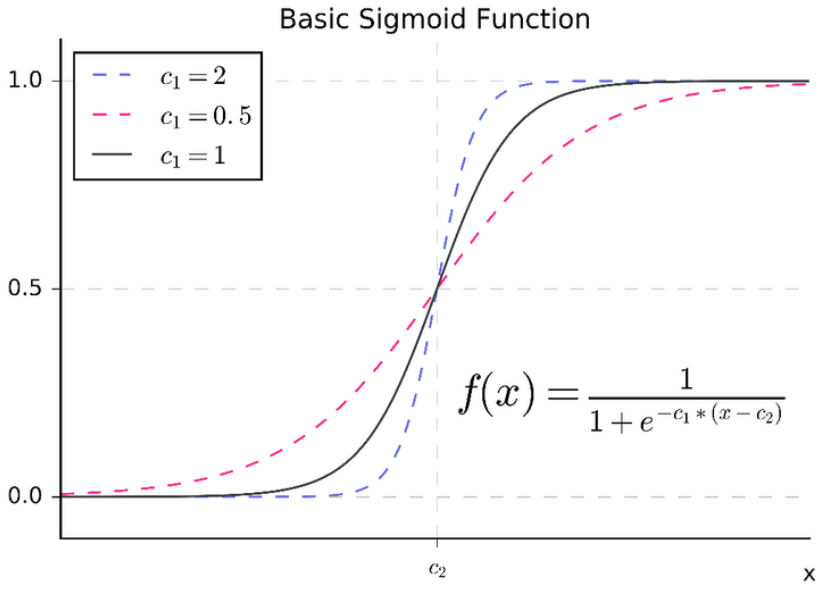


Figure 2.1: The logistic sigmoid function [22].

using machine learning, we apply Occam's razor, or, as often called, the principle of parsimony [21]. What is meant by that, is the fact, that rather than trying high-end machine learning models, simple machine learning are tested to see how they perform.

Logistic regression

One of the oldest classification models, Logistic regression, is named after the use of a sigmoid (logistic) function

$$\sigma(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x}^T \cdot \mathbf{w}}}, \tag{2.1}$$

where $\mathbf{x}$ stands for vector of input features and $\mathbf{w}$ stands for vector of weights. If we look at the image of function σ (figure 2.1, we can see that values are in range $(0, 1)$. This means that the probabilty of predicted class in a binary classification task can be defined as follows

$$P(y = 0|\mathbf{x}) = \sigma(\mathbf{x}, \mathbf{w}), \tag{2.2}$$

$$P(y = 1|\mathbf{x}) = 1 - \sigma(\mathbf{x}, \mathbf{w}), \tag{2.3}$$

where $y \in \{0, 1\}$ is predicted label.

Optimal weights $\mathbf{w}$ are found minimizing a selected loss function (more in section 2.5.3).

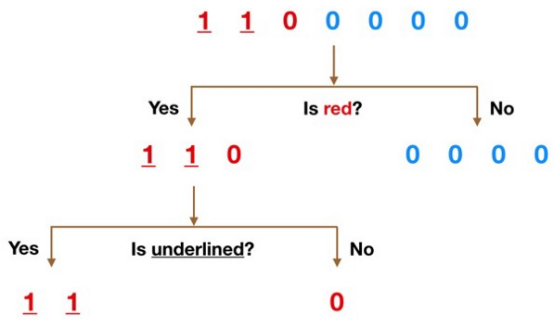


Figure 2.2: Binary decision tree functionality illustrated [26]

Random forest

A binary decision tree is a structure based on a subsequent decision process made by asking a series of questions. Starting from the root of the tree, a feature is evaluated and, depending on the outcome, one of the two branches is selected to proceed further. This procedure is repeated until a leaf of the tree is reached [23]. This leads to the binary decision tree being one of the most intuitive, as, instead of having to observe calculated weights, one can simply look at the "questions asked".

To acquire such questions, first, a dataset distribution has to be measured and suitably divided. This can be achieved using Gini index, which is a summary statistic measuring the equability of dataset distribution. Considering the dataset is ideally split, the split receives a Gini score of 0. Otherwise, values range from 0 to 0.5, where 0.5 stands for the worst split[5]. After splitting the dataset in numerous ways and evaluating all splits by a chosen criterion function (in this case, Gini index), the best performing split is chosen as the node in the tree [25]. Starting from the root, all nodes are calculated until maximmum tree depth is achieved. Maximum tree depth is a hyperparameter stating number of nodes from the root node of the tree to the leaves. The higher value of tree depth, the higher chance of overfitting [25].

To obtain a higher performance, binary trees are used in ensembles, meaning, many trees are used and their predictions are averaged for the final output value, helping the overall stability of the classification. However, this can only be achieved, if each tree is created based on a different dataset, done by dividing the whole training set into smaller sets [26]. One of the algorithms for dividing training sets is called Bootstrap aggregating. It, first, generates m uniform training subsets with replacements, meaning, a fraction of the dataset consists of repeating values. This helps random forests tremendously,

[5]The calculating process of the Gini index can be seen at [24].

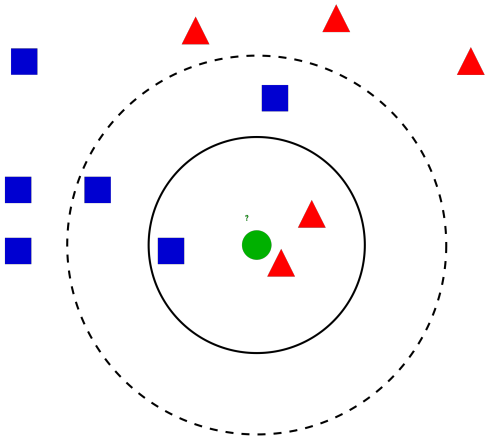


Figure 2.3: k-NN classifier illustration. For $k = 3$, l_p is red triangle, for $k = 5$, l_p is blue square [28].

as even a small change in the dataset can result in a very different decision tree [27].

k-Nearest Neighbors algorithm

Another model known for its simplicity and intuitivity. Suppose n rows with m continuous features exist. First, all of the rows are placed in $\mathbb{R}^m$ space. Let

$$C_i = (c_{i1}, \ldots, c_{im}) \in \mathbb{R}^m, \ i \in \{1, \ldots, n\} \tag{2.4}$$

be coordinates of i-th row in the training set. Then $c_{ij}, \ j \in \{1, \ldots, m\}$ stands for j-th feature of the i-th row. Also, let

$$l_i \in \{0, 1\}, i \in \{1, \ldots, n\} \tag{2.5}$$

be the label value for i-th element in the set.

Predicting label l_p of an input with feature values resulting in coordinates C_p starts with ordering all C_i coordinates by a selected norm:

$$||C_p - C_1|| \leq \cdots \leq ||C_p - C_m||, \tag{2.6}$$

resulting in $(C_1, \ldots, C_k)$ being $k \in \mathbb{N}$ nearest neighbors. Label value with the highest amount of appearances in $(l_1, \ldots, l_k)$ becomes the predicted label l_p. The process is also illustrated in figure 2.3.

2.5.2 Neural network models

With the increase of computational power, neural networks recently gained a lot of popularity. Two types are especially important, as others are often built, at least partly, from them - fully connected neural nets and convolutional neural nets [29].

Fully connected layers

Fully connected layers are made of layers of neurons, often described in comparison to the neurons of a human brain. Each neuron in a layer n takes each value passed from previous layer $n-1$, $x_{n-1,i}$, multiplies it with the neurons learned weight w_i and adds bias b [29]:

$$y_{n,j}(\mathbf{x}_{n-1}) = f(\mathbf{w}_j \cdot \mathbf{x}_{n-1} + b), \tag{2.7}$$

where $y_{n,j}$ stands for output of j-th neuron in n-th layer and $f(x)$ stands for non-linear activation layer. Often times, we write the equation 2.7 as matrix multiplication:

$$\mathbf{y}_n(\mathbf{x}_{n-1}) = f(\mathbf{W} \cdot \mathbf{x}_{n-1} + \mathbf{b}),\ \mathbf{W} = (\mathbf{w_1}, \ldots, \mathbf{w_J})^T \in \mathbb{R}^{I \times J}, \tag{2.8}$$

where I stands for number of neurons in layer $n-1$ and J for number of neurons in layer n. The situation is illustrated in figure 2.4.

Convolutional layers

Convolutional layers introduce sliding dot product kernel of weights. Now, only of partion of neurons from previous layer impacts neurons in the next layer, however, thanks to this, number of learnable weights is a lot smaller, making the network a lot more flexible, and also, much more observable, especially in case of classificating images [29].

Process of calculating the output values of a convolutional layer is illustrated in figure 2.5. Many parameters can be set, in respect to the kernel. Size determines number of weights in each kernel. Number of kernels is set by the output dimension of convolutional layer. Stride stands for the size of increment in each sliding of the kernel.

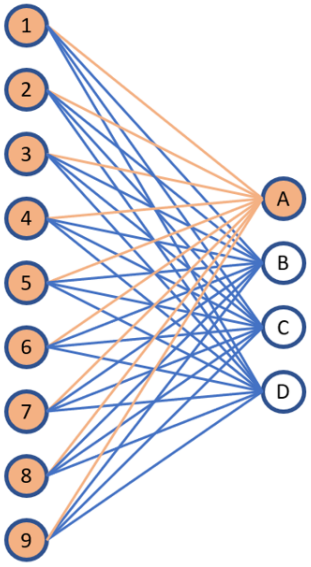


Figure 2.4: Fully connected layer illustration [29]

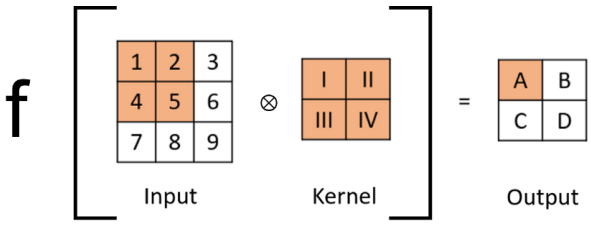


Figure 2.5: Dot product of convolutional layer [29]

Embedding layer

Embedding layer is a popular way of working with discrete-valued features in neural nets.

Embedding layer's number of weights $\mathbf{W}$ is determined by the count of unique discrete values ps (pool size) passed to the layer, and a hyperparameter called embedding dimension d. The unique values are, first, assigned indices.

Using these indices, one-hot encoded vectors are created:

$$\mathbf{v} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{\text{ps}}. \tag{2.9}$$

The whole process of calculating the output values for two one-hot vectors looks as follows:

$$\mathbf{Y} = \mathbf{W} \cdot \begin{bmatrix} v_1 & v_2 \end{bmatrix}, \ \mathbf{Y} \in \mathbb{R}^{d \times 2}, \ \mathbf{W} \in \mathbb{R}^{d \times \text{ps}}$$

Optimizers

Having a loss function allows for performing the backpropagation algorithm for calculating derivation of the loss function with respect to the input values given. With partial derivations of all learnable weights, many algorithms are able to start the process of tuning weights. We call these algorithms optimizers [33].

Gradient descent is one of the oldest optimizers, and also very intuitive. Let $\mathbf{w}_n$, $\mathbf{w}_{n+1}$ be current weight values and weight values after the next step gradient descent, then

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma \cdot \nabla \mathbf{L}(\mathbf{w}_n), \tag{2.10}$$

where γ is a learning rate and $\nabla \mathbf{L}$ a gradient of loss function. The only parameter passed to the algorithm is the learning rate. Choosing a learning rate might be very tricky part of the learning process. High learning rate leads to overstepping (illustrated in figure 2.6), while low learning rate might lead to finding a local minimum and being unable to step out of it.

Many other advanced algorithms exist to make the learning process faster and more robust. In this study, Adam, in other words, adaptive moment estimation algorithm is used as optimizer. It utilizes the momentum by adding fractions of previously calculated gradients to the current one [33].

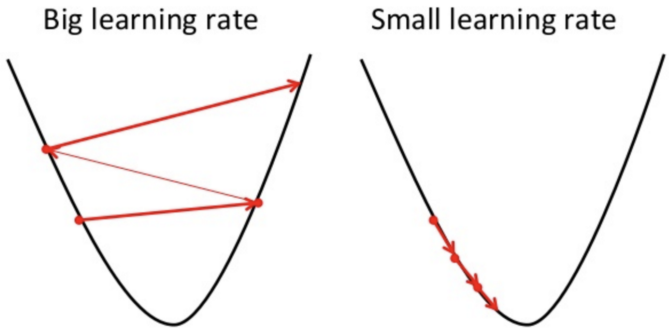


Figure 2.6: Illustration of too big of a learning rate resulting in overstepping the minimum [33].

2.5.3 Loss function

Directly calculating optimal weights of machine learning models is, in most cases, with currently discovered tools, an impossible task, once number of weights exceeds a certain amount. Instead, the problem is essentially narrowed to an optimization task of finding the lowest value of a loss function by navigating n-dimensional space (n stands for number of learnable parameters - weights). The loss function, sometimes called cost function, is used to evaluate how well the predicted values of a model match the ground truth labels [31]. It reduces all aspects of a possibly complex system down to a single continuous scalar, which allows to compare found partial solutions. Due to that, it is very important to choose a loss function well fitting to our goals [30].

Cross-entropy loss

In the context of binary classification, cross-entropy loss function, also called logarithmic loss, or logistic loss, is very popular. Cross-entropy loss value is calculated this way:

$$L(y, \hat{y}) = -y \log \hat{y} + (1 - y) \log(1 - \hat{y}), \tag{2.11}$$

where $y \in \{0, 1\}$ is the actual label value and $\hat{y} \in [0; 1]$ is the predicted value. It can be observed, that depending on the value of an actual label, either first, or second part of the expression is used [31].

As we can see in figure 2.7, small differences of predicted and true labels result in really small loss function values, meanwhile, large differences result in a really high loss function values.

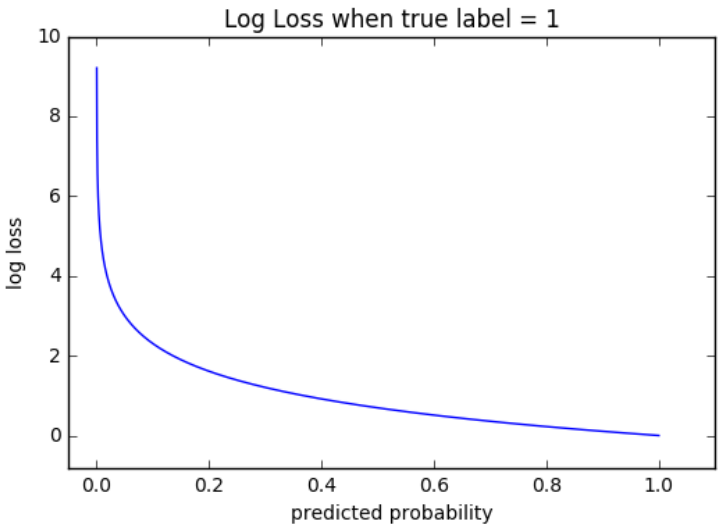


Figure 2.7: Cross-entropy loss function image for y = 1 [32]

Chapter 3

Data collection

Data collection process navigates the content of this chapter. It starts with researching available options in terms of datasets. After that, process of data scraping, and tools used to reach certain goals, are described. Chapter is ended by section regarding database structure designing.

3.1 Data research

While no official application containing data of competitive matches exists, there is a few options in terms of third party applications.

Sites providing API services are posing a very convenient way to obtain large chunks of data. For a considerable fee, data are given within hand's reach as there is no need to program any scraper to start working. There is a trade off of getting only a fraction of features that can be obtained in other ways.

First of the two sites in table above that require scraping is called Wewatch. It provides complete match feed (rounds and kills in chronological order), meaning most of the features that are otherwise provided by other applications could be potentially computed after the scrape. Potentially new features could also be computed afterwards. Unfortunately, Wewatch only provides a very small dataset of matches.

	HLTV	Panda Score	Game Score keeper	Wewatch	Kaggle dataset
Number of matches	$\sim$ 65,000	?	?	$<$ 1000	$\sim$ 25,000
Match feed	✗	✗	✗	✓	✗
Round results	✓	✓	✗	✓	✓
Round economy	✓	✗	✗	✓	✓
Detailed player stats	✓	✓	✗	✓	✓
Vote-ban	✓	✗	✓	✗	✓
Kill matrix	✓	✗	✗	✓	✗
GOTV demos	✓	✗	✗	✗	✗
Heatmaps	✓	✗	✗	✗	✗
API	✗	✓	✓	✗	
Price	✗	$150 per month	?	✗	✗

Table 3.1: Comparison of dataset sources

HLTV is Counter-Strike's household name. It offers data of every match of any importance in the competitive scene of Counter-Strike: Global Offensive. Not only is the dataset quite large, HLTV also shows most of the features, that can be obtained about a match. Beside the usual features such as round results and detailed player stats for each map, it also offers features such as vote-ban phase, round economy data, GOTV demos, event phase (groups, playoffs, ...), information about whether a match was played on LAN or online, player kill matrices and more. Sadly, some of the features, such as round economy, are only present in matches played since various dates (discussed further in 3.2). HLTV also provides no API and data, often presented on multiple URL's at the time, requiring many HTTP requests, need to be obtained via scraping.

Kaggle dataset posted by a user called @mateusdmachado[1] poses as an interesting alternative. Its data are scraped from HLTV and hold approximately 25,000 matches dating from November, 2015 to March, 2020. The only downside is that around 15,000 matches have been played since March, 2020, meaning a scraping is still the preferred way.

[1]Dataset URL: https://www.kaggle.com/mateusdmachado/csgo-professional-matches

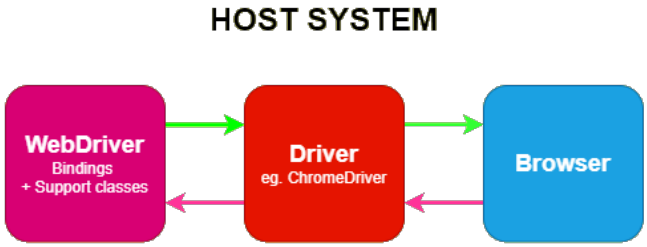


Figure 3.1: Direct WebDriver-Browser communication diagram [35]

3.2 Data scraping

3.2.1 Data scraping software

After careful considerations, it has been decided to go the way of scraping the HLTV site's data. Regarding scraping software, there is a few options to choose from.

First one to come in mind is Scrapy. It is a web crawling framework, used to scrape websites and extract structured parsed data [34]. One of Scrapy's biggest advantages is being easy to pick up and scalable at the same time. However, it lacks any advanced tools for working with data exchanged by a scraped server and client's browser (such as cookies). Scrapy pretty much only offers to change the User-Agent request header, meaning, if the scraped server puts up any effort in recognizing the difference between a real browser and a scraping tool, Scrapy is very likely to fail.

That is where Selenium WebDriver comes in. Selenium WebDriver handles a browser natively, as a real human user would, either locally, or on a remote machine [35]. Selenium goes a step further to act as a human user to the outside world. "At its minimum, WebDriver talks to a browser through a driver. Communication is two way: WebDriver passes commands to the browser through the driver, and receives information back via the same route." [35] The driver (the red part of the diagram in figure 3.1) is specific to the browser used to scraping. Thanks to this layout, setting various cookies and also loading site's JavaScript scripts is possible.

Figure 3.2: HLTV's results page

3.2.2 HLTV site structure

To get URL links to each of the matches of the dataset, one has to first enter the "Results" part of the site via the navigation bar button (number 1 in figure 3.2). Each results page holds exactly one hundred matches (match number range shown next to number 2 in the same figure; page elements referring to individual matches as number 3). The simplest way to obtain links to all of approx. 64,000 matches is to cycle through results pages using generated URL addresses `https://www.hltv.org/results?offset=i`, where $i \in \{0, 100, ..., 64000\}$ and saving the `href` attribute value of each match element.

Entering match page, a number of potential features is shown. Match information (text field, number 1 in figure 3.3) contains information about whether the match has been played Online or on LAN and the series format. Next, there is text field showing vote-ban sequence (number 2 in the same figure). The main match page also offers a link to the GOTV demo files[2] of the played maps (number 4). However, as stated before, HLTV presents match features (and especially individual map features) spread out in multiple pages entered via "STATS" buttons (number 3).

First such page is called "Overview" (fig. 3.4), where a round history of the played map is shown (number 1 in the figure), as well as detailed player statistics collected at the end of the map (table, number 2 in the figure).

[2]GOTV demos are files containing every action happening in a match. Anyone having CS:GO installed on their computer can run GOTV demo to receive lossless rerun of the recorded match.

Figure 3.3: HLTV's Match Page layout

Both are expected to be key features in future experiments. While the player statistics give somewhat accurate idea of which players had the highest impact on the match outcome, round history describes the development of the map. What makes round history especially interesting, is the number of possibilities, in terms of features, it generates (especially, when combined with map economy data - figure 3.5).

"Economy" page gives complementary data to overview's round history. The significance of the economy to the game has already been described in detail in chapter 2.3. Money spent by a team in a round is saved as css title attribute of an image corresponding to the round and team.

Finally, there is the "Heatmaps" subsection (figure 3.6). It allows generating heatmaps for very specific filters (number 1 in the figure) - filtering individual players, kills, deaths, play sides. Unfortunately, each generated heatmap also requires a new HTTP request, meaning, collecting the heatmaps easily pumps up the number of requests per map from 3 to 80, or even 160.

The other feature in this subsection is hidden in roll-down menus (number 2). It contains data about individual guns used by each player to kill. For example, this feature can give an idea of which player performs well in pistol rounds, or using sniper rifles.

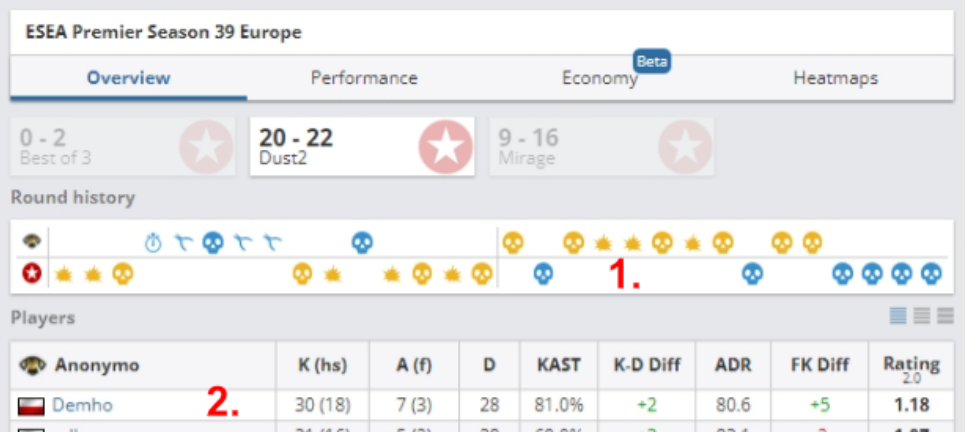


Figure 3.4: HLTV's Map Overview Page layout

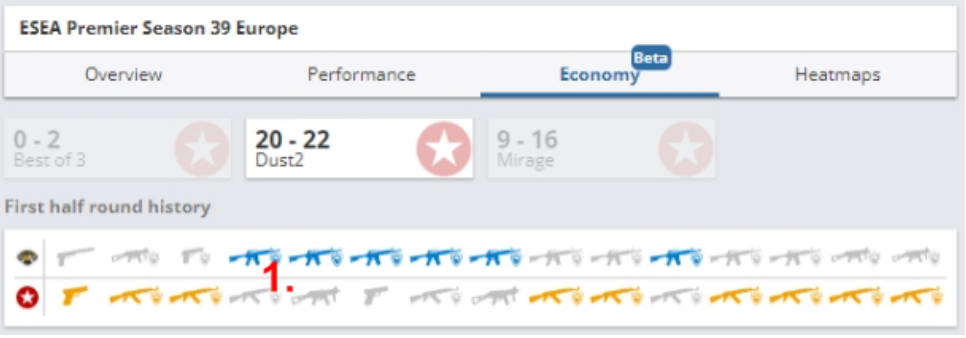


Figure 3.5: HLTV's Map Economy Page layout

3.2.3 Scraping process

HLTV's anti-scraping protection at the beginning of this project's data collection (namely February 2021) allowed full scraping using Scrapy, which made selection of scraping software very convenient. At first, the data were parsed using Scrapy and immediately saved into database. That allowed for very quick data collection, however, at a cost. Should the wanted list of features change, the whole scraping process would have to be repeated. To avoid that, it was later decided to perform a two-step scraping process (figure 3.7), where all HTML requests are first scraped and saved as whole HTML files, that are parsed into database in the second step. This proved to be very helpful as it improved project's scalability and also helped uncovering data hidden in the CSS attributes.

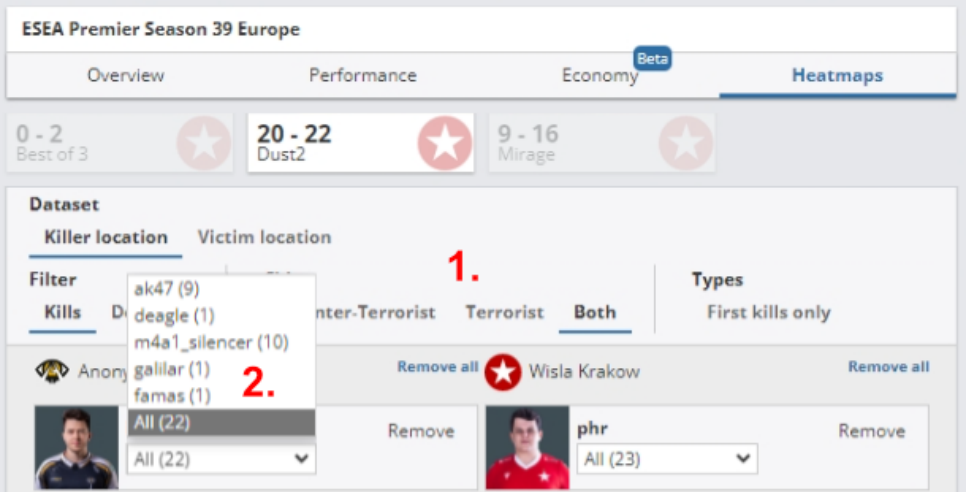


Figure 3.6: HLTV's Map Heatmap Page layout

Figure 3.7: Scraping process visualized

Files are saved following this hierarchy:

```
data_<results offset>
├── missed_pages.txt
├── matches
│   └── <date>
│       └── <time>_<match ID>
│           └── <map number>_<map ID>_<map name>
│               ├── overview.html
│               ├── heatmap.html
│               └── economy.html
```

The `missed_pages.txt` file stores URL's of all requests that ended with an unhandled error, as throughout the year, HLTV changed its anti-scraping protection, leading to certain pages being unscrapable using Scrapy.

3.2.4 Parallelization of the scraping process

With the ambition to scrape heatmap data, resulting in 80+ requests per map, and the site only allowing a minimum delay of 2-3 seconds, it became apparent that parallelization of the process would be needed.

The first solution to come in mind is using proxy servers as using proxy servers is natively supported by Scrapy. That would allow executing each script on the same computer. Unfortunately, proxy servers offered for free turned out to be very unstable and unreliable to be of any use, and paid proxy server services usually allow for significantly lower number of requests per month (in a reasonable price range). For example, the ScraperAPI proxy servers offered specifically for scraping cost $249 per month and allow up to 3 M requests. Total amount of requests needed for the whole scraping process is around 10 M requests.

Next in line is the use of distributed computing infrastructure such as MetaCentrum Virtual Organization. It consists of computing and storage resources owned by CESNET and is free for researchers and students of academic institutions in the Czech Republic. Its two main services are grid and cloud computing. Grid computing uses multiple distributed resources, while Cloud computing is a method of sharing resources, such as storage space. For the parallelization part, the grid services are certainly more fitting, while cloud services suit the parsing task better.

MetaCentrum VO Grid services

Using grid services is all about creating jobs and passing them to a scheduler that assigns resources based on number of various variables (duration of the job, needed CPU and RAM resources, ...). Running a job can be done in two ways - either by creating a batch script or interactively. Running an interactive job suits best for figuring out a working script to pass to scheduler in batches. A job's key argument passed to scheduler is cluster wanted by the client to run the job. Using different clusters works exactly as running the scraper on different computers.

3.3 Database structure

After running all HTTP requests and storing the HTML files, data need to be parsed and saved to a database. There is a number of database engines to choose from. Due to the fact, that during the work on this project, many computers were used, an appropriate engine would be ideally plug-and-play and also allow for moving data quickly. Sqlite is very lightweight database engine that store all database data in a single file, that fit the needs of this

project, hence chosen.

The key entities represented by individual tables in the database would be:

- match,
- map,
- round,
- roster,
- player.

`Match`, as an entity, allows to look at all the data in the context of time, and so, in the context of other matches. It allows for the main entity, `Map`, to take date and time of happening, number in the match series, recognize whether match was played online or on LAN event. Some of the improvements, such as storing the event phase data and also the addition of veto phase, would also be tied to `Match`.

`Map` is possibly the most important entity of the whole project. The same way `Match` allows `Map` to take place in time, `Map` allows grouping and ordering of `Round` instances, linking `Roster` instances and individual `PlayerStats`. Each instance of `Map` is also linked to a table named `MapName` via an ID, allowing for filtering data by each individual level in the game.

`Round` holds the data used for computing majority of team features. Each `Round` instance corresponds to a single `Roster` instance (via `MapID` and `RosterNumber`), meaning, there are two instances in the table for each in-game round. It also contains information about playing side and round result. Both of these ID's are links to different tables `PlaySide` and `RoundResult`.

Each `Match` instance spawns a new pair of `Roster` instances, meaning, same roster can have multiple instances with different ID's. `RosterHash` is calculated from sorted `PlayerID`'s of its players and serves the purpose of linking the same rosters together.

`PlayerStats` tie `Map` and `Player` instances to offer the player features. Each `PlayerStats` instance can also relate to many `WeaponKills` instances.

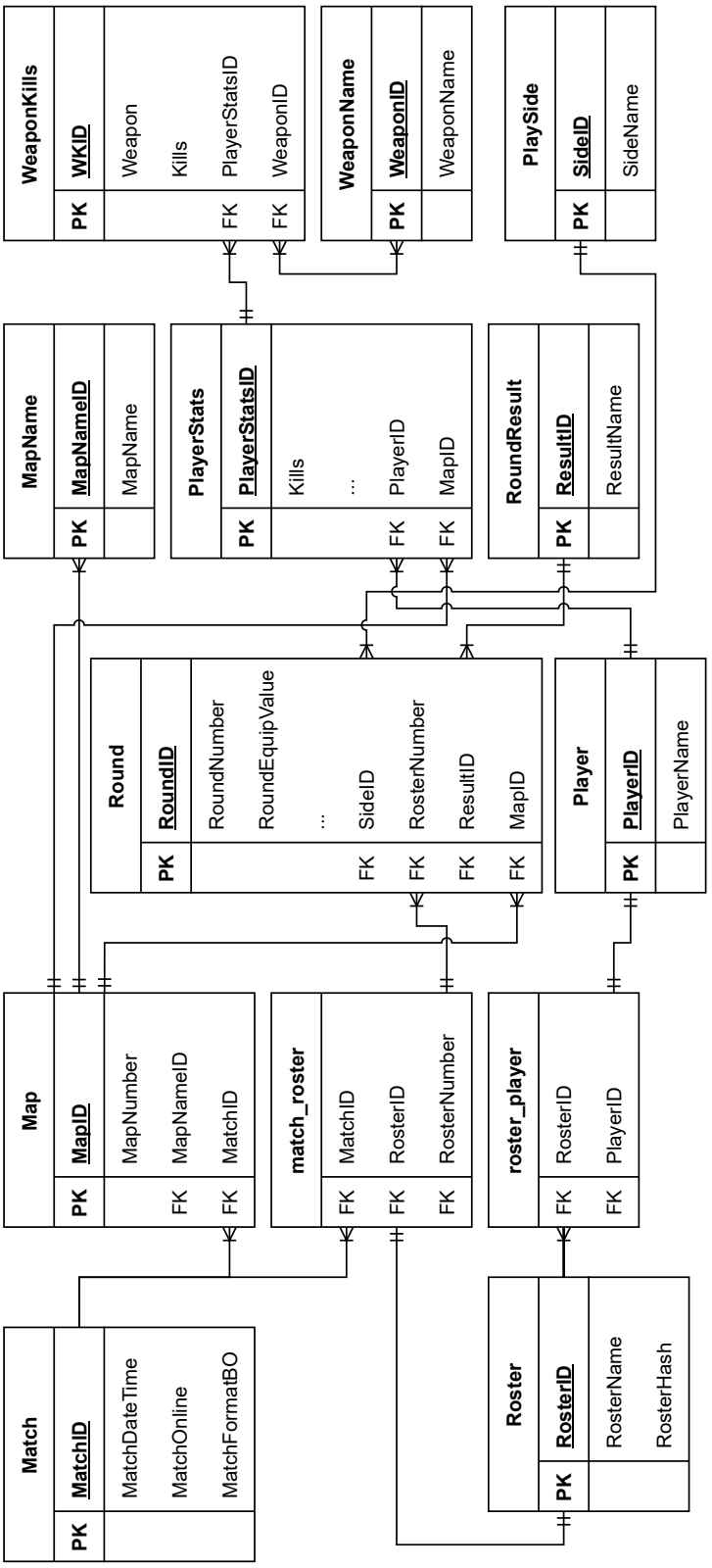


Figure 3.8: Database layout

Chapter 4

Data analysis

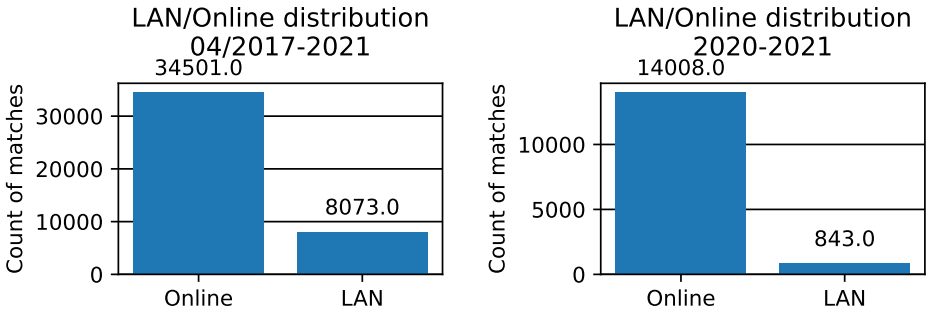
4.1 Contextual statistics

Now, with data scraped and parsed in database, a time has come to take a look at the data. In this section, an effort is spent to give context to the data and offer an explanation to various variables and phenomena.

4.1.1 Matches

In regards of matches, we can tell what BO format is each match, and whether it has been played online or on LAN. As we can see in figure 4.1a, approximately four fifths of scraped matches played since April 2017 to 2021 were played online. It makes perfect sense considering that many lower-tier tournaments are played online, as the costs of running LAN tournaments grow high, both for teams that spend money travelling to the area, and organizers hosting the event. What has helped skewing data the most, is Covid-19 global pandemic. In the figure 4.1b, we can see, that since the start of 2020, only very small fraction of matches were played on LAN, showing how big of an impact it had on the sport.

In relation to predicting of match outcomes, knowing if a match is played online can, in specific cases, play a role. Those would occur, when players



(a) : Counts of matches played Online and on LAN throughout the whole dataset

(b) : Counts of matches played Online and on LAN throughout the Covid-19 epidemic

Figure 4.1: Comparison of LAN played portions of dataset in relation to Covid-19 pandemic

with no history of LAN play in the match, as no experience with playing in front of an audience tend to cause additional stress, especially, considering younger players. Another such case would be a newly assembled roster, with no prepared tactics, relying on in-game skill, that tends to drop when under stress of playing on stage.

Figure 4.2 shows, that the vast majority of matches is played in either BO1 or BO3 format. BO1 matches are usually taken less seriously, than BO3 matches. It is broadly believed, that the more maps in a series, the lower chance of an underdog winning it. However, it is yet to be proven, if there actually is consistency difference in different BO formats in CS:GO.

BO2 matches are only used at league format tournaments. Leagues are very unusual in CS:GO compared to, let's say, League of Legends, where majority of professional matches are played in leagues. Thus, such a small representation in the dataset.

4.1.2 Maps

We can see (figure 4.3), that some maps are being played a lot more often than others. Two of the most played maps, Mirage and Inferno, have been played about 5,000 more times than the next most played map. That is not entirely caused by being players' favorites, as it could seem at first glance, but also competitive map pool (list of maps featured in competitive environment;

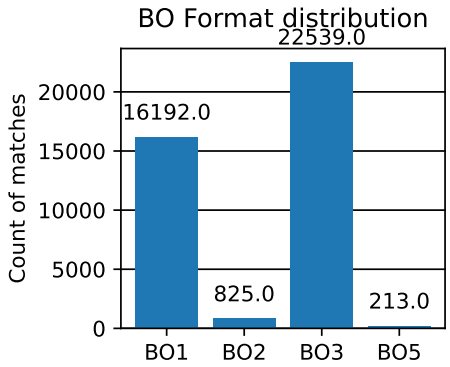


Figure 4.2: Count of matches played in different BO formats

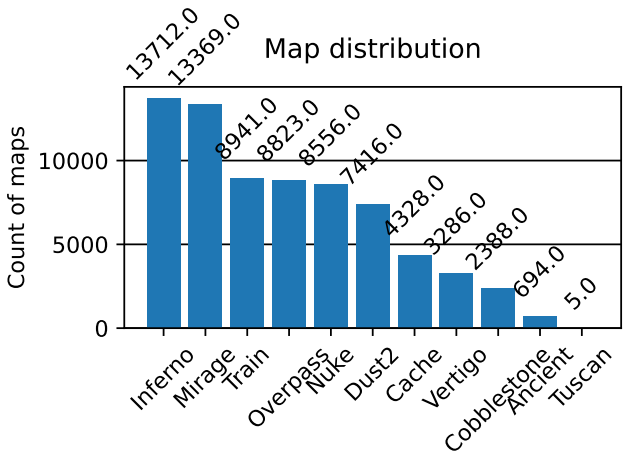


Figure 4.3: Counts of map matches played divided by in-game levels

also called the active duty group) rotations made by the developers of the game. Some maps have entered the active duty group recently, for example Ancient (released May 3, 2021), some have been out of the group for months, due to an ongoing development of revised versions, other maps have been put out of the group and never placed back, such as Cobblestone (visualized in figure 4.4).

For predicting, it is important to take into consideration, that a newly revised map is often very different, in terms of tactics and gameplay, to its predecessor, and should be treated as a different map.

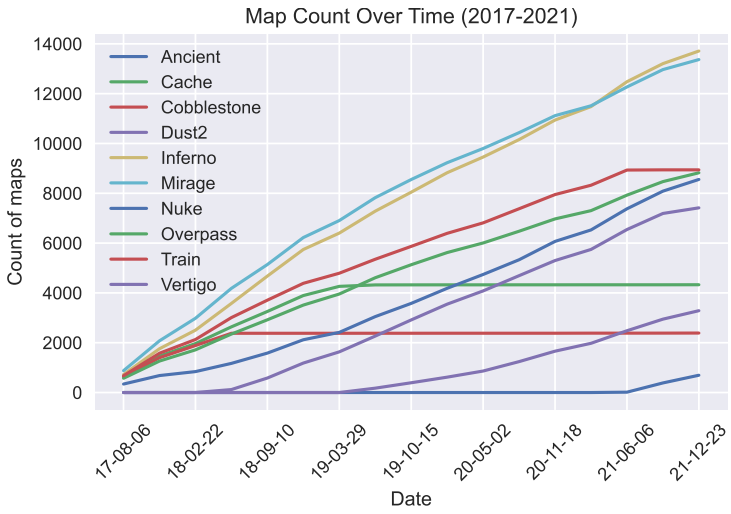


Figure 4.4: Map played counts over time

4.1.3 Rosters

Predicting outcomes could possibly benefit from knowing how long have the playing teams been playing with the same roster. Roster changes can help freshen up and 'rekindle a spark' to the long lasting teams. On the other hand, teams usually achieve in-depth tactics after spending longer amounts of time together.

Considering the whole scraped dataset, teams average 24 days together before making a roster change. However, half of the dataset is made of teams having only played 1 match. After removing such teams, we get mean of 53 days and median of 22 days. Even after removing teams with 1 match (0 days), half of the dataset is made of teams lasting under 22 days. That is not a long time. Perhaps, this feature could be helpful when only evaluating top tier teams, as in case of the best teams, players are usually held by signed contracts with organizations.

4.1.4 Weaponry

Currently, there is six weapon classes in the game, four of which could be of importance - rifles, snipers, pistols and SMG's. From figure 4.5, it can be observed, that kills with rifles are in significant majority.

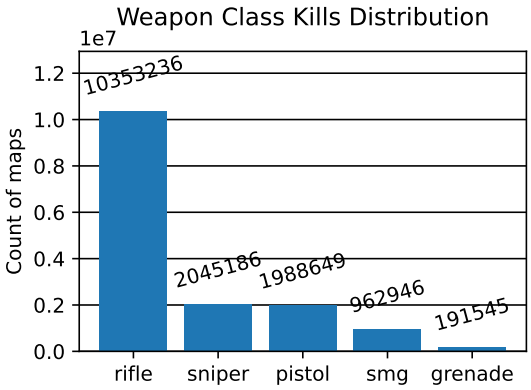


Figure 4.5: Weapon Class Kills Distribution

By itself, kills with rifles have no value. Every single player in the game gets kills with rifles. However, it can be used as benchmark for other classes. For example, ratio of sniper kills to rifle kills helps recognize AWP players (AWP is an in-game sniper rifle). While predicting round to round outcomes, ratio of pistol and SMG kills to rifle kills might help recognize teams with strong economy rounds (rounds with low equipment value bought).

4.2 Feature selection

In this study, entirely all features used for predicting are made of in-game features, with an exception of calculated elo rating.

4.3 Turning data into features

This section goes through making features suitable for acting as input for predicting models. To achieve that, two popular Python libraries, Pandas and NumPy, are used.

4.3.1 Player features

The goal is to get player feature values divided by number of rounds a player has played up to a certain point. Usually, players' statistics are used in "per map" context, however, in case of predicting CS:GO outcomes, it is important to take into account, that some maps only last 16 rounds, while others last up to 30 rounds, meaning, stats from some maps carry more weight than from others.

At the beginning, table `Map` is merged with table `Round` to gain the number of rounds played for each map, resulting in a new Pandas DataFrame `MapWithRoundCount`. Merging the newly created DataFrame with table `PlayerStats` leaves us with a DataFrame containing all players statistics with rounds played on the side, `map_playerstats_dfs['all']`.

The next step is to calculate cumulative sums and perform dividing with cumulative sum of round count:

```
map_playerstats_dfs['all'] = map_playerstats_dfs['all'].join(
    map_playerstats_dfs['all'].groupby('PlayerID').agg({
        'Kills': np.cumsum,
        'Headshots': np.cumsum,
        [...]
        'RoundCount': np.cumsum,
    }).add_suffix('_cum'))

for col in stats_to_round:
    map_playerstats_dfs['all'][col + 'PerRoundExpanding'] =
    map_playerstats_dfs[df_name][col + '_cum']
```

Listing 4.1: Calculating of player stats' cumulative sums

Process is, then, repated for all the maps separately in different DataFrames.

4.3.2 Roster features

Roster features make more sense expressed in percentages. For example, instead of "EcoRoundsWonPerRound" it makes more sense to have "EcoRoundsWonPercent".

The process starts with summing `Round` table grouped by `MapID` and `RosterNumber`:

```
rndf = rndf.groupby(['MapID', 'RosterNumber']).agg({
    'TeamWonRound': np.sum,
    'TeamCtWin': np.sum,
    'TeamTWin': np.sum,
    [..],
    'RoundsCount': np.size
}).reset_index()
```

Listing 4.2: Calculating sums of round features

Next, `Round` DataFrame is merged with `Roster` DataFrame to allow for calculating of cumulative sum grouped by `RosterHash`. Lastly, percentages are calculated:

```
map_teamstats_dfs['all'] = map_teamstats_dfs['all'].join(
    map_teamstats_dfs['all'].groupby('RosterHash').agg({
    'TeamWonRound': np.cumsum,
    'TeamCtWin': np.cumsum,
    'TeamTWin': np.cumsum,
    [...],
    'RoundsCount': np.cumsum,
}).add_suffix('_cum'))

for perc_col, (round_col, win_col) in perc_stats_cols.items():
    map_teamstats_dfs['all'][perc_col] = \
        map_teamstats_dfs['all'][win_col] / map_teamstats_dfs['
    all'][round_col]
map_teamstats_dfs['all']['TeamTWinPerc'] = \
    map_teamstats_dfs['all']['TeamTWin_cum'] / (
                map_teamstats_dfs['all']['RoundsCount_cum'] -
    map_teamstats_dfs['all']['TeamCtPlayed_cum'])
```

Listing 4.3: Calculating of player stats' cumulative sums

Once again, the process is repeated for all maps separately in different DataFrames.

4.4 Missing data

There are many ways to treat missing data in a dataset. Essentially, there are two options. Either deleting rows with missing values, or imputing the values in some way.

Deleting rows with missing values, while being the simplest option, carries two risks. First, it can create bias in data, if dropped data are not randomly

distributed. Secondly, it can result in losing large amounts of data, drastically decreasing dataset size.

Imputing missing values with one selected value per feature is another convenient way to treat missing data. Usually, data are imputed either with mean or median. This approach is running a risk of losing covariance between features.

In this study, both approaches are tested, to see whether one dominates the other, and also combined.

4.4.1 Dropping maps with no economy data

Specific case of missing data in the dataset is presented by missing economy statistics in all of the maps before a given date. Due to this fact, all the maps before April 28, 2017 are dropped from the dataset, as it makes little to no sense to impute two years of continuous missing data. This results in dropping 14,954 rows of maps, leaving a total amount of 72,247 rows to analyse and predict from.

Chapter 5

Experimentation process

5.1 Working with dataset

Throughout the experimenting part of this study, data from the dataset are split into three categories. This approach aims to prevent models from overfitting by splitting the training process into three phases.

5.1.1 Training set

Models, accompanied by chosen optimizers and loss functions, use the training dataset to set the learnable parameters. Usually, a model is passed the training set in cycles over and over again. Each iteration of the cycle is called an epoch. As a sign of successful fitting process, we expect the loss over training dataset (training loss) to keep dropping to zero each epoch.

5.1.2 Validation set

While it is nice to see the training loss drop to zero, it can (and, in most cases, does) mean that the model is overfitting and will not perform when

passed unknown values.

To successfully recognize the moment a model starts overfitting the training dataset, each epoch we also evaluate the value of loss function over the validation dataset (test loss), however, this time without calculating backward propagation. At first, validation loss should be dropping similarly to the training loss. Once validation loss starts raising, it is a sign that model is starting to overfit.

5.1.3 Testing set

Last step of evaluating the performance of a model is to pass it completely new data from the reserved testing set. This is to ensure that the design of the model is not overfitted to the validation set.

5.1.4 Growing window

Ideally, each time an outcome is predicted for a map, a model would be trained on precisely all the data in the dataset that chronologically happened before the map in mind.

To accelerate the process, some of the accuracy is sacrificed and the training dataset is grown by chunks of data. In other traditional sports, each chunk would be a single season.

As previously mentioned, CS:GO is not running in seasonal league format. Instead, heaps of tournaments by different tournament organizers from all over the world are hosted. Each year, a tournament called major is played. It is a huge tournament overseen by the developers themselves, and carries a large amount of prestige. Usually teams practice all season to measure forces at major events. Not everyone can win, though, which means some teams usually disband. This makes ends of the major events good places to split individual chunks of data.

Dataset sizes for the growing window algorithm are shown in the table 5.1 below.

Major event name	End date of training set	End date of validate set	Number of rows in training	Number of rows in validate
EL Major 2018	2018-01-19	2018-01-28	7,735	291
StarSeries 4	2018-02-17	2018-02-25	8,399	353
FaceIt Major 2018	2018-09-12	2018-09-23	17,973	355
IEM Katowice 2019	2019-02-20	2019-03-03	23,683	390
ESL One Cologne 2019	2019-07-02	2019-07-07	29,093	119

Table 5.1: Growing window row sizes

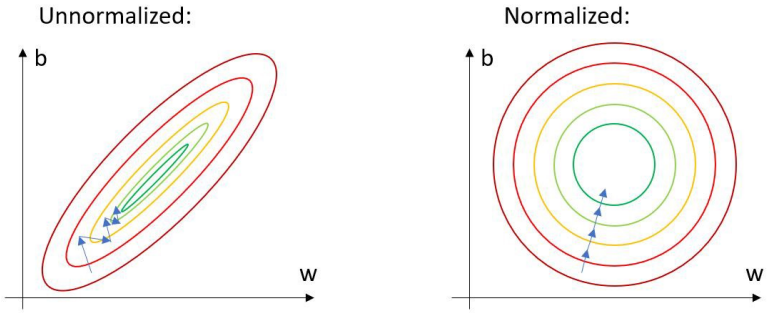


Figure 5.1: Comparation of gradient descent in unnormalized and normalized datasets (image taken from [36])

5.1.5 Data preprocessing

All features used in this study, besides the roster hash and player ID features, are continuous. Different features' values have different distributions. Some features' values range from 0 to 100, some range from 0 to 1. This leads to problems with learning via gradient descent. The reason is illustrated in the figure 5.1 below, where gradient descent is shown. In the left portion of the picture, steepest gradient is found, yet, due to the features' varying ranges, optimizer overshoots each step, resulting in oscillation.

Hence, each feature is normalized by, first, subtracting a calculated mean from the values, and then, by dividing the values by calculated standard deviation:

$$\text{new\_value} = (\text{old\_value} - \mu)/\sigma \tag{5.1}$$

This is very effectively achieved by using `StandardScaler`, a class from scikit-learn Python library. Following code, first, calculates means and standard deviations by executing `StandardScaler`'s `fit` function. In this case, both training and validation sets are used for calculations. Next, values are normalized by calling the `transform` function on each set separately.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

sc.fit(np.concatenate((X_train, X_test), axis=0))
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

Listing 5.1: Using StandardScaler to perform data preprocessing

5.1.6 Sample representations

Data are represented by roster and player features calculated from all previously played maps, across all map levels, and feature calculated from previously played maps on the same map level.

In this thesis, three separate sample representations are tested - player features representation only, roster features representation only, and then the concatenation of both.

5.2 Implemented models

This study tests non-neural network models and different designs of neural network models, aiming to figure out best ways for predicting match outcomes. Proposed neural network models use combinations of linear, convolutional and embedding layers, and compare their performances to other selected machine learning models, as well as a selected baseline.

5.2.1 Using non-neural machine learning models

Choosing the same models as previously described in the Background chapter, we end up with three non-neural network machine learning models:

- Logistic Regression,
- Random Forest classifier,
- k-Nearest Neighbors classifier.

First in the list, the logistic regression, is implemented from module `sklearn.linear_model`. The second listed model, the Random Forest classifier is implemented from module `sklearn.ensemble`. Lastly, k-NN classifier is implemented from module `sklearn.neighbors`. For these models, calling a function `fit` once is enough for the process to be finished. Example of usage is shown in listing 5.2.

```
models = [
    linear_model.LogisticRegression(parameters_lr),
    ensemble.RandomForestClassifier(parameters_rfc),
    neighbors.KNeighborsClassifier(parameters_knn),
]

for model in models:
    model.fit(X_train, y_train, sample_weight=sample_weights)
```

Listing 5.2: Using non-NN machine learning models

5.2.2 Using the linear model

Linear model is instance of class `basic_model/NeuralNetwork` that inherits from PyTorch's `nn.Module`. An instance is created by calling:

```
model = basic_model.NeuralNetwork(
    file_name=file_path_string, # path model is loaded/saved
    from
    epoch_count=EPOCH_COUNT, # integer value
    learning_rate=LR, # float value
    dropout=dropout_percentage, # float value from 0 to 1
    batch_size=50, # integer value
    ts_size=len(TS_cols)//2, # integer value
    ps_size=len(PS_cols)//2, # integer value
    linear_base=128
```

```
)
```

Listing 5.3: Creating linear neural network class instance

Where `ts_size` stands for number of roster features per roster and `ps_size` for number of player features per roster.

Model is trained by calling function `trainf`:

```
model.trainf(X_train, y_train,
             X_test, y_test,
             weights_train=sample_weights, # bool
             shuffle_train=False
)
```

Listing 5.4: Calling training function of linear model

Where `X_train` stands for NumPy matrix of dimensions $m \times n$, where m is number of rows in train dataset and n is number of all features combined. `y_train` is NumPy vector of length m. Same goes for parameters `X_test` and `y_test`. Sample weights are a vector of length m. They are used for weighing samples of the training dataset.

Order of features passed is not important to this model as long, as it's kept same as used in training.

5.2.3 Using the convolutional model

Convolutional model is instance of class `conv_model/NeuralNetwork`, inheriting from `nn.Module`. An instance of model is created by executing:

```
model = conv_model.NeuralNetwork(
    file_name=file_path_string,  # string
    epoch_count=EPOCH_COUNT,  # integer
    learning_rate=LR,  # float
    dropout=dropout_percentage,  # float 0 to 1
    batch_size=50,  # integer
    ts_size=len(TS_cols)//2,  # integer
    ps_size=len(PS_cols)//10,  # integer
    linear_base=32,  # integer
    conv_depth=64  # integer
)
```

Listing 5.5: Creating convolutional neural network class instance

There are two differences to creating the linear model. One is, that parameter `ps_size` is passed as number of features per single player. The other is `conv_depth`, that is used in the previous section 6.1.2.

The `trainf` function is passed same arguments as in the case of linear model. The only difference is, that this time the order of features matters. First `ts_size` of features are expected to be first roster's team statistics, then 5×`ps_size` of first roster's player variables, followed by the same amount of features of the second team in the same order.

5.2.4 Using of the embedding model

Same as both previously mentioned models, embedding model's class `embedding_model/NeuralNetwork` is also a child of the `nn.Module`. Constructor is called this way:

```
model = embedding_model.NeuralNetwork(
    file_name=file_path_string,  # string
    epoch_count=EPOCH_COUNT,  # integer
    learning_rate=LR,  # float
    embedding_dim=10,
    roster_pool=roster_pool,
    player_pool=None,
    weights=weights,
    ts_size=len(TS_cols)//2,
    ps_size=len(PS_cols)//2,
    linear_base=64,
    batch_size=20
)
```

Listing 5.6: Creating embedding neural network class instance

In this case, passing of either roster or player pool is needed for correct initialization of the model. One of the two values is expected to be None and the other a NumPy vector containing unique identifiers of all rosters/players (hash/ID). `embedding_dim` stands for a hyperparameter characteristic to embedding layers. `ps_size` is passed same way it is in the case of the linear model.

The `trainf` function used for training the model requires four new parameters. Let's say there is n_1 samples in the training dataseta and n_2 samples in the testing dataset. The `RH_train` (`PID_train`) and `RH_test` (`PID_test`) should then have n_1 and n_2 rows respectively.

```
model.trainf(X_train, y_train,
```

```
2                 X_test, y_test,
3                 RH_train, PID_train,
4                 RH_test, PID_test,
5                 weights_train=sample_weights,
6                 shuffle_train=False
7 )
```

Listing 5.7: Calling training function of embedding model

Chapter 6

Results

6.1 Chosen architectures and hyperparameters of different models

Hyperparameters were found out using a method, were each model only barely manages to overfit the training dataset.

6.1.1 Linear model

Two different architectures were tested with different sets of parameters. Specifically, a pyramid architecture and architecture used now were tested. The theory behind a pyramid architecture, where each layer further from inputs is smaller, is that data about the input get more condensed and proper representation of the input for outputting the required class is achieved. In this case, however, different types of architectures performed very similarily and the second architecture was selected, performing a tiny bit better.

Linear model used in this study consists of four hidden linear layers. The hidden layers have symmetric dimensions. Dimensions of all hidden linear

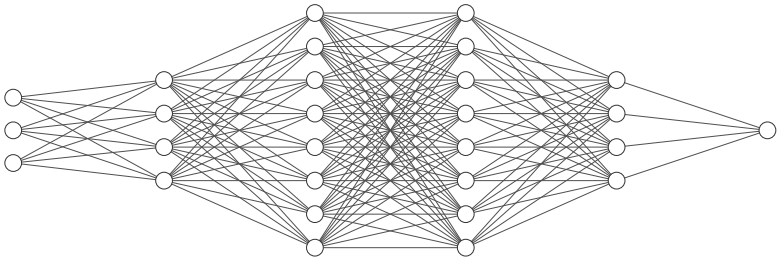


Figure 6.1: Linear layers of linear model

layers are hyperparameters set by model's attribute called `linear_base`:

$$\text{dimension of outer layers} = \texttt{linear\_base},$$
$$\text{dimension of inner layers} = 2 \times \texttt{linear\_base}.$$

`linear_base` used in the experiments equaled the number of input features.

In the figure 6.1, outer hidden layers are represented by layers with 4 nodes, inner hidden by layers with 8 nodes. The first layer, consisting of three nodes, is an input layer, while the last one (to the right) is an output layer.

Each layer, besides the output layer, is accompanied by batch norm, dropout and activation (Tanh) layer. Dropout is another hyperparameter, set by model's attribute `dropout`. As default value, dropout is set to 50 %. The output layer has Sigmoid activation on the output.

6.1.2 Convolutional model

Convolutional neural network designed in this thesis inherits linear layers from the previous linear model. Difference is, that player features are not directly passed to the input linear layer. Instead, they are passed to three convolutional layers.

In the first convolutional layer, i one-dimensional filters of length `ps_size` and stride `ps_size` iterate through player features of a single team. This results in $5 \times 1 \times i$ values. Next layers divides depth by 2. Third convolutional layer has k one-dimensional filter of length 2, stride 1 and depth j. $4 \times 1 \times k$ values are flattened into a single vector and concatenated to roster features.

The i, j, k values are defined by model's parameter `conv_depth`:

$$i = \texttt{conv\_depth},$$
$$j = \frac{\texttt{conv\_depth}}{2},$$
$$k = \frac{\texttt{conv\_depth}}{4}.$$

Throughout the experiments, values of `linear_base` = 32, `conv_depth` = 64 performed the best.

6.1.3 Embedding model

Embedding model is pretty much the same as linear model with one exception. Values of hashes (or ID's in case of player embedding) are first mapped to their indices, passed to an embedding layer and output is then concatenated to the input features.

`embedding_dim` of value 10 was used.

6.2 Evaluation of learned models

In this section, all learned models' performances are evaluated based on their accuracies and compared to the selected baseline. The evaluating process consists of comparing two values for each model and window size.

The first criterion is models' average validation accuracy for different sample representations. To calculate this average accuracy, accuracy over validation set with imputed missing values and accuracy over validation set with dropped missing values (trained on training sets with same methods of data treatment).

The second criterion is models' average validation accuracy for datasets with dropped and imputed missing values over different sample representations.

Based on these criteria, one model from neural networks and one non-neural

machine learning model is chosen and test loss is calculated to give this study an output.

6.2.1 Baseline model

In this thesis, one baseline model is used for comparison to the selected models. The baseline model only predicts the first team to win. While there is no clear reason why, as no home advantage exists in Counter-Strike, and HLTV does not inform users of placing teams with higher ranking in the left side, all dataset splits are skewed to the left (`roster_number` = 0 in the database) creating an impression that some underlying system of placing teams with higher chance of winning to the left is, indeed, used. It will be often referred to as the T1 baseline for short.

6.2.2 Evaluation of non-neural network models

All three models show no difference in validation accuracy for different treatment of missing values in the dataset. The reason was not found and more investigation needs to be done in the future.

Logistic regression's accuracy started quite high with the first two windows. It managed to reach over the 60 % accuracy mark with all three sample representations, with player and combined representation sets beating the Elo model. With growing training set, accuracy of logistic's regression trained on team representation plummeted to 52 %, below the T1 baseline model. Still, two other sample representations managed to keep accuracy over 60 %, though, below the Elo ranking prediction model.

k-Nearest Neighbors model never really managed to get over the 60 % accuracy mark, only barely beating the T1 baseline. kNN model proved to be unfitting of the task.

Random forest's performance proved to be the most consistent one. Even with team-based sample representation only, random forest managed to beat the T1 baseline, as the only one from non-neuron machine learning models selected.

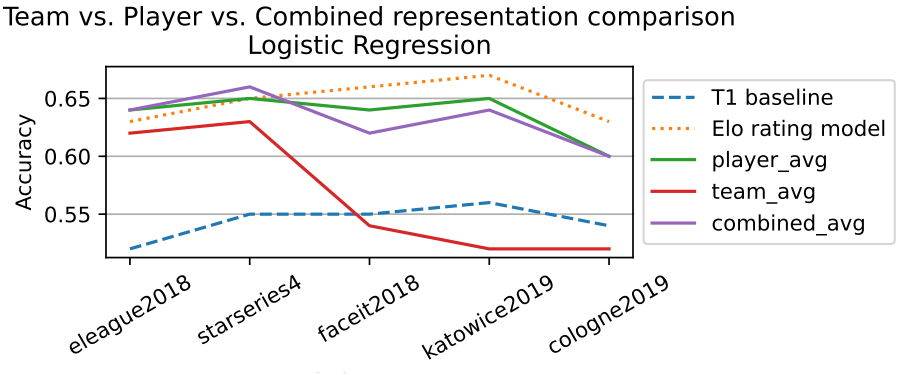


Figure 6.2: Logistic regression accuracy based on sample representation

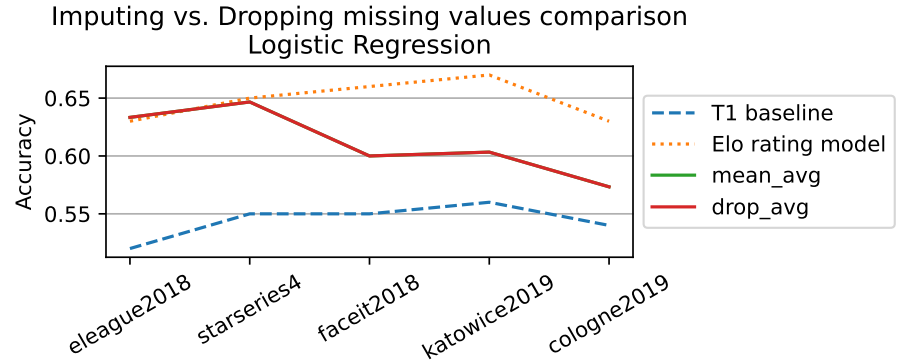


Figure 6.3: Logistic regression accuracy based on missing data management

Due to that, random forest is chosen as the model to be tested further on the test set.

6.2.3 Evaluation of neural network models

Linear model with fully connected layers managed to keep up with Elo ranking model pretty well, however, not to beat it. Team-based sample representation experienced a drop, though, not as bad as it did with model using embedding layer.

There is a phenomenon observable from figures 6.9 and 6.11 (marginally, also from 6.13), where accuracies of neural network models start higher for training sets with imputed values and, as higher volume of data appear in the training set, dropping missing values starts being beneficial, as there is presumably enough data to sacrifice some of it.

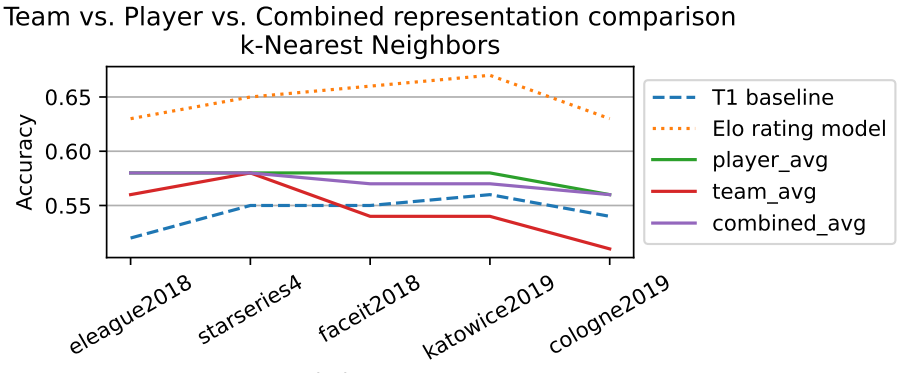


Figure 6.4: k-Nearest Neighbors accuracy based on sample representation

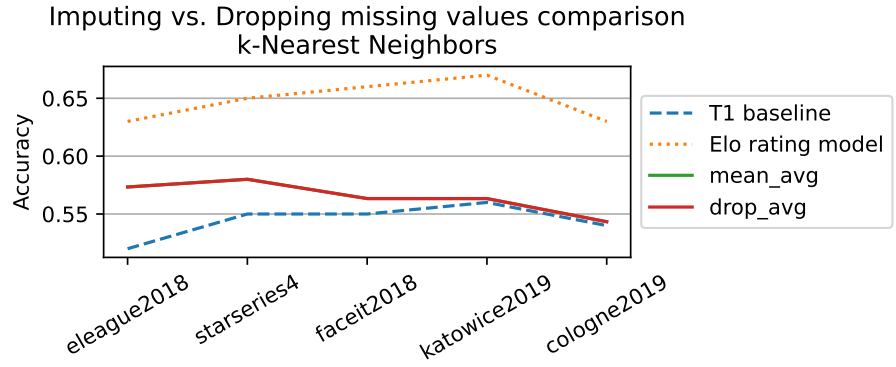


Figure 6.5: k-Nearest Neighbors accuracy based on missing data management

Convolutional neural network, of course, doesn't have averages calculated for team-based sample representations, as its main functionality revolves around working with player's statistics and player representations. Similar results to the linear model were measured.

All things considered, embedding model did pretty good, comparing it to non-neural network models. However, it is not really consistent with its accuracy jumping up and down each window (figure 6.12.

Convolutional neural network is selected for calculating the test precision.

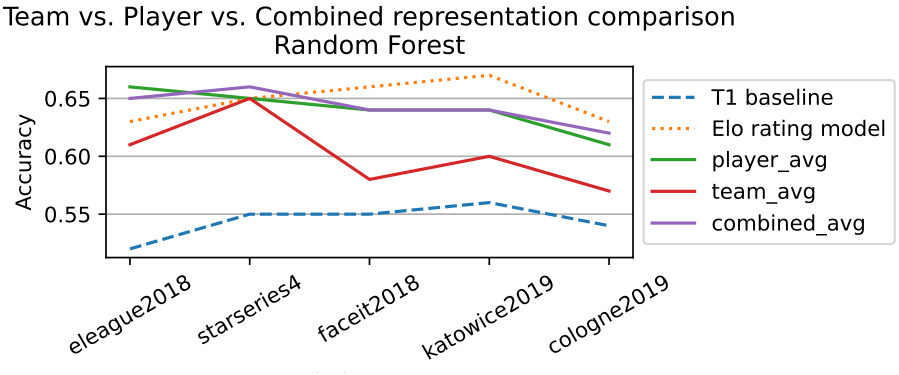


Figure 6.6: Random Forest accuracy based on sample representation

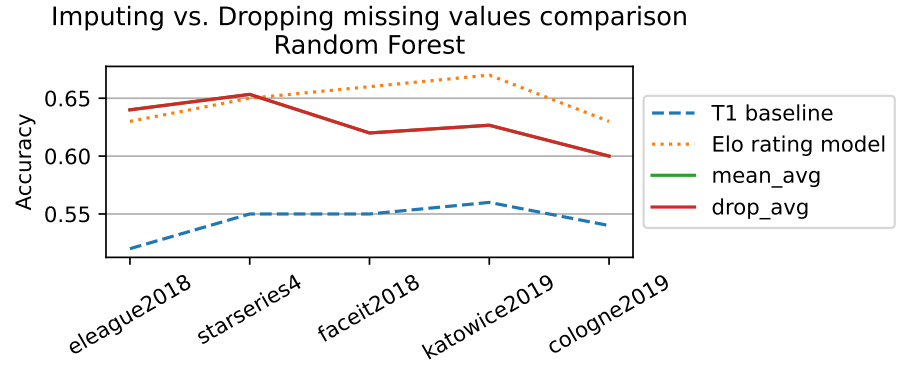


Figure 6.7: Random Forest accuracy based on missing data management

6.3 Test set evaluation

Left out test set is made of 28,766 maps to predict. Random Forest, reaching a 99.8 % accuracy on the training set with combined sample representation and dropped missing values, managed to predict 63.0 % of the test set outcomes right. Convolutional neural network managed to learn on the same dataset for 20 epochs before valid loss started rising. It got test accuracy of 59.8 %.

Model	Missing data	Sample representation	Training set Accuracy	Test set Accuracy
Random Forest	Dropped	Combined (team + player)	99.8 %	63.0 %
CNN	Dropped	Combined (team + player)	66.4 %	59.8 %
T1 baseline				55.0 %
Elo-based model				64.0 %

Table 6.1: Accuracy values obtained by predicting the test set

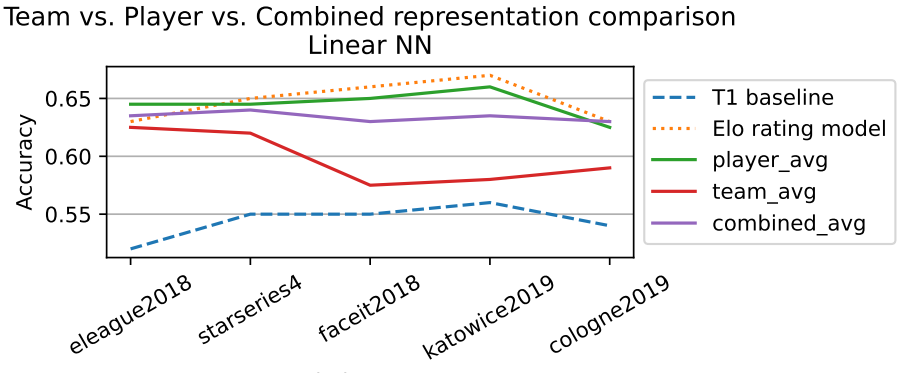


Figure 6.8: Linear model performances on different sample representations.

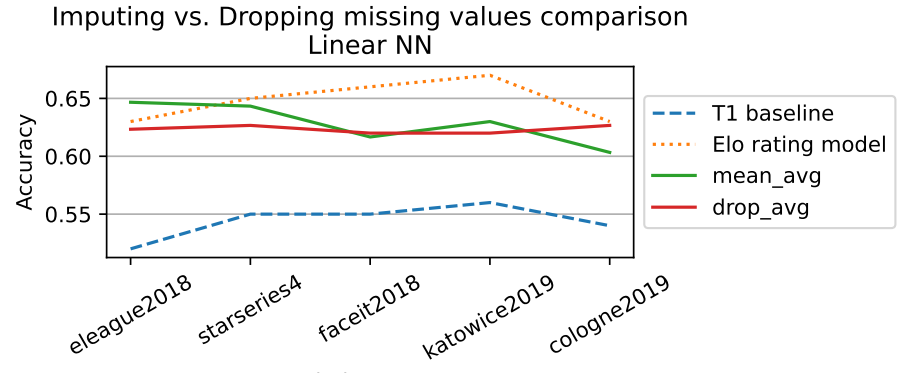


Figure 6.9: Linear model performances with different preprocessing methods.

6.4 Feature importance

With Scikit's Random Forest Classificator having native attribute of feature importances, that can be plotted, it makes sense to observe, what features managed to make a difference.

Using the MDI method described in Scikit's documentation, we plot feature importances in figure 6.14. Only features with mean decrease in impurity of value equal to 0.12 and higher are shown, to make the plot a bit cleaner.

Unsuprisingly, elo averages are one of the features showcased, as elo consistently dominated all selected models throughout the thesis. Other than that, KD ratios and, also, separately kills and deaths show as important features, which makes sense, considering the nature of the game.

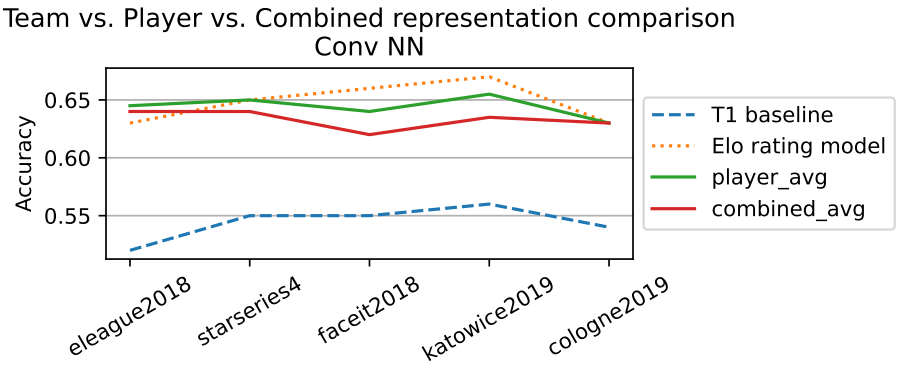


Figure 6.10: Convolutional model performances on different sample representations.

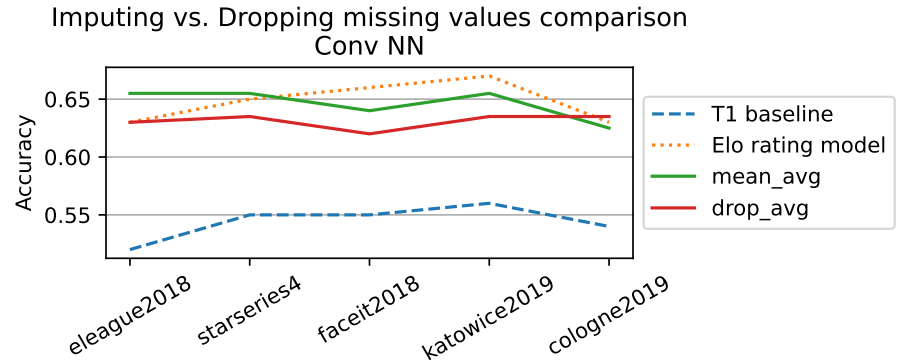


Figure 6.11: Convolutional model performances with different preprocessing methods.

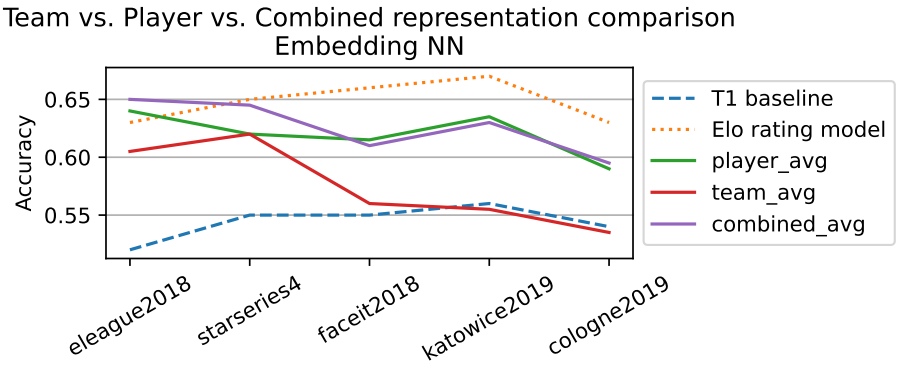


Figure 6.12: Embedding model performances on different sample representations.

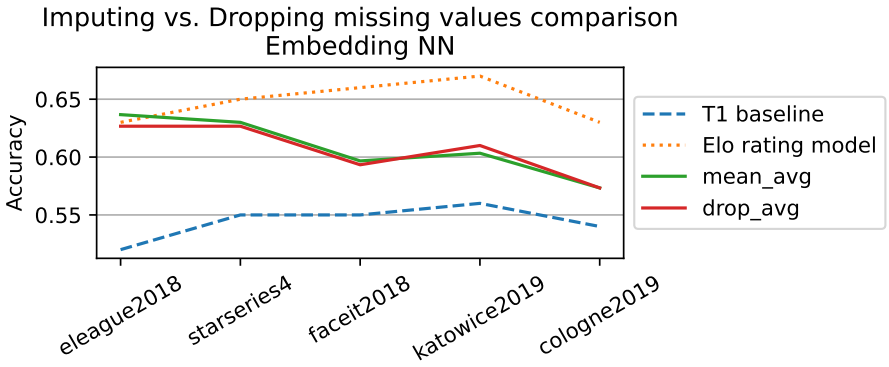


Figure 6.13: Embedding model performances with different preprocessing methods.

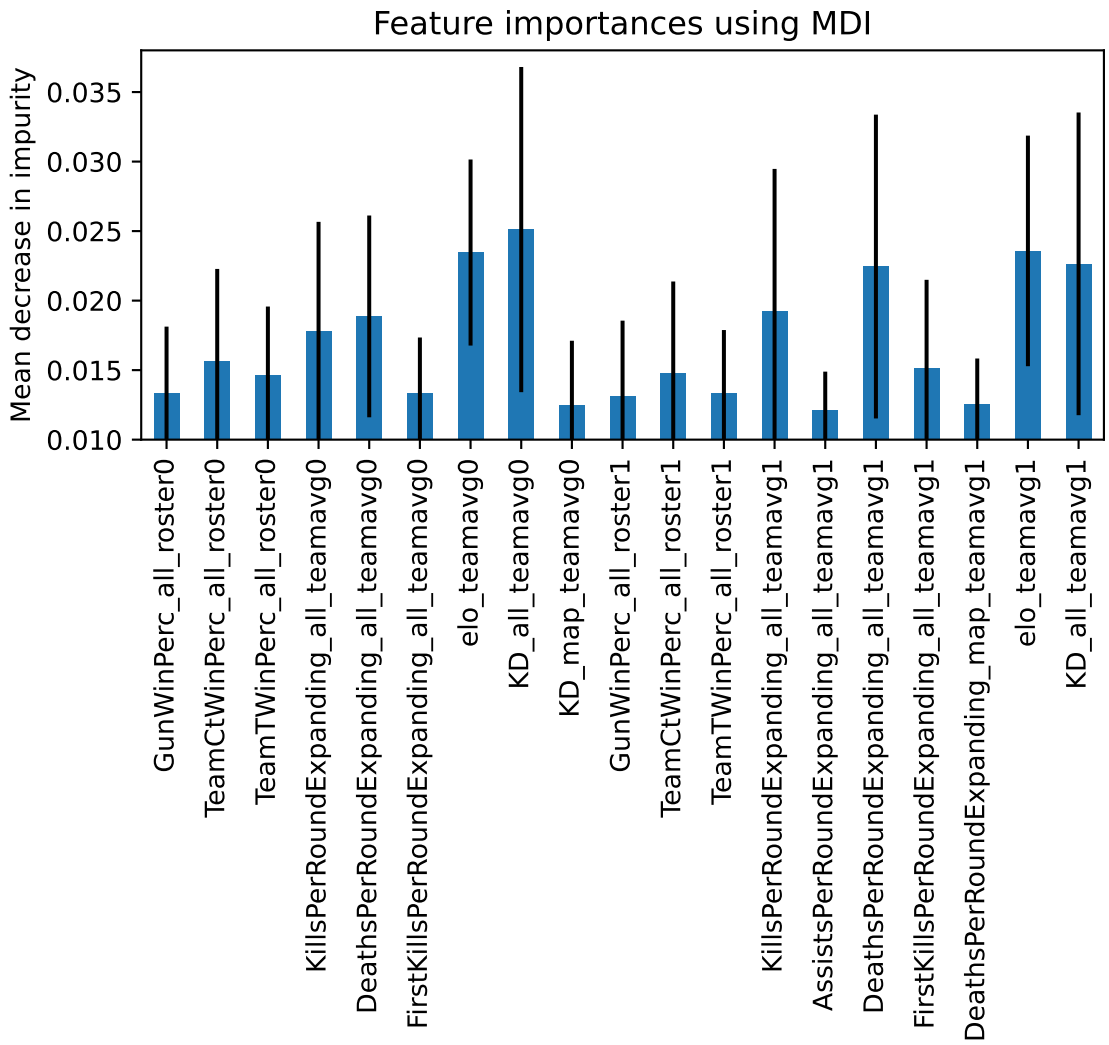


Figure 6.14: Feature importances

Chapter 7

Conclusion

This thesis' goal was to perform an exploratory machine learning study regarding outcome prediction in professional matches of CS:GO.

In pursue of this goal, first, a dataset of substantial volume had been scraped and stored in a database specifically designed to allow for future work and scalability, exceeding any other datasets freely available, both, in size, and features offered.

Next, in chapter 4, data were closely analyzed in order to gain a better understanding of the problem. Three sample representations were designed, based on this newly acquired knowledge. One consisting exclusively of player features, one consisting of team features, and one as combination of the two approaches. To tackle the problem of predicting outcomes with features developing in time, an expanding window view of the data has been utilized in the cumulative feature creation process in order to respect the natural chronological order of the data.

Three standard statistical and three neural network machine learning models were used to perform a series of testing. These machine learning models are directly compared to a model based on Elo ratings of players, and a baseline only predicting one side based on the observed data skewness. Accuracy of 64,0 % has been achieved with the best performing model based on the Elo rating of players, followed by 63,0 % accuracy with the random forest model and 59.8 % accuracy with the convolutional neural network, all beating the baseline model with 55.0 % accuracy. Compared to the 62.0 %

accuracy reported in a previous work [17] on a significantly smaller dataset, two of our models managed to achieve better predictive accuracy.

7.1 Future improvement

Although beating the selected baseline, the Elo rating being the highest ranking model signalizes a lot of room for improvement. Seeing Elo perform so well, a question arises whether some better rating-based models, such as TrueSkill 2 or Gecko ratings, could perform even better.

Moreover, the predictions might also profit from other sample representations based on features like tournament phase, vote-ban phase, map number in a series, previous head-to-head team performances, and such, providing further context to each match.

Appendix A

Bibliography

[1] BELLIS, Mary. A Brief History of Sports: From Rocks and Spears to Laser Tag. ThoughtCo [online]. 2019 [cit. 2022-01-04]. Available from: https://www.thoughtco.com/history-of-sports-1992447

[2] SCALETTA, Kelly. Settling the Debates: Is It a Sport or Not a Sport? Bleacher Report [online]. 2011 [cit. 2022-01-04]. Available from: https://bleacherreport.com/articles/848465-settling-the-debates-is-it-a-sport-or-not-a-sport

[3] PwC's Sports Survey 2018 [online]. 2018 [cit. 2022-01-04]. Available from: https://www.pwc.ch/en/insights/sport/sports-survey-2018.html

[4] PwC's Sports Survey 2019 [online]. 2019 [cit. 2022-01-04]. Available from: https://www.pwc.ch/en/insights/sport/sports-survey-2019.html

[5] PwC's Sports Survey 2020 [online]. 2020 [cit. 2022-01-04]. Available from: https://www.pwc.ch/en/insights/sport/sports-survey-2020.html

[6] ESports market revenue worldwide from 2019 to 2024. Statista [online]. 2021 [cit. 2022-01-04]. Available from: https://www.statista.com/statistics/490522/global-esports-market-revenue/

[7] Global Esports Live Streaming Market Report [online]. Newzoo, 2021 [cit. 2022-01-04]. Available from: https://resources.newzoo.com/hubfs/Reports/2021_Free_Global_Esports_and_Streaming_Market_Report_EN.pdf

[8] GAINSBURY, Sally. Editorial. International Gambling Studies [online]. 2010, 10(1), 1-4 [cit. 2022-01-04]. ISSN 1445-9795. Available from: doi:10.1080/14459791003760882

[9] TURCU, I., G.B. BURCEA a D.L. DIACONESCU. THE IMPACT OF THE BETTING INDUSTRY ON SPORTS. In: Series IX Sciences of Human Kinetics [online]. 2020, s. 251-258 [cit. 2022-01-04]. ISSN 23442026. Available from: doi:10.31926/but.shk.2020.13.62.2.32

[10] BLOCK, Sebastian, Florian HAACK a T. KLIESTIK. ESports: a new industry. In: SHS Web of Conferences [online]. 2021 [cit. 2022-01-04]. ISSN 2261-2424. Available from: doi:10.1051/shsconf/20219204002

[11] SteamCharts: An ongoing analysis of Steam's concurrent players [online]. [cit. 2022-01-04]. Available from: https://steamcharts.com/app/730All

[12] SHENG, Albert. As esports grows, so too do its sponsorships. WIN.gg [online]. [cit. 2022-01-04]. Available from: https://win.gg/news/as-esports-grows-so-too-do-its-sponsorships/

[13] 1xBet becomes Official Global Betting Partner for ESL Pro Tour CS:GO and ESL One Dota 2 [online]. 2021 [cit. 2022-01-04]. Available from: https://about.eslgaming.com/blog/2021/03/1xbet-becomes-official-global-betting-partner-for-esl-pro-tour-csgo-and-esl-one-dota-2/

[14] FIALHO, Gabriel, Aline MANHÃES a João Paulo TEIXEIRA. Predicting Sports Results with Artificial Intelligence – A Proposal Framework for Soccer Games. Procedia Computer Science [online]. 2019, 164, 131-136 [cit. 2022-01-04]. ISSN 18770509. Available from: doi:10.1016/j.procs.2019.12.164

[15] KHANTEYMOORI, Alireza a Davoodi ELNAZ. Horse racing prediction using artificial neural networks [online]. 2010 [cit. 2022-01-04]. Available from: https://www.researchgate.net/publication/228847950_Horse_racing_prediction_using_artificial_neural_networks

[16] PEARCE, Tim a Jun ZHU. Counter-Strike Deathmatch with Large-Scale Behavioural Cloning. CoRR [online]. 2021 [cit. 2022-01-04]. Available from: https://arxiv.org/abs/2104.04258

[17] MAKAROV, Ilya, Dmitry SAVOSTYANOV, Boris LITVYAKOV a Dmitry I. IGNATOV. Predicting Winning Team and Probabilistic Ratings in "Dota 2" and "Counter-Strike: Global Offensive" Video Games. Analysis of Images, Social Networks and Texts [online]. Cham: Springer International Publishing, 2018, 2018-12-21, , 183-196 [cit. 2022-01-04]. Lecture Notes in Computer Science. ISBN 978-3-319-73012-7. Available from: doi:10.1007/978-3-319-73013-4_17

[18] BJÖRKLUND, Arvid. Predicting the outcome of CS:GO games using machine learning [online]. Gothenburg, Sweden, 2018 [cit. 2022-01-04]. Available from: https://hdl.handle.net/20.500.12380/256129. Bachelor Thesis. Chalmers University of Technology / Department of Computer Science and Engineering (Chalmers).

[19] HOEKSTRA, Kyle. Who Was Chess Master Arpad Elo, and What is the Elo Rating System? History Hit [online]. 2021 [cit. 2022-01-04]. Available from: https://www.historyhit.com/gaming/arpad-elo-rating-system/

[20] MADDEN, Ryan. Adapting Elo for Relative Player Ranking in Team-Based Games [online]. 2017 [cit. 2022-01-04]. Available from: https://ryanmadden.net/posts/Adapting-Elo

[21] DUIGNAN, Brian. Occam's razor. Britannica [online]. 2018 [cit. 2022-01-04]. Available from: https://www.britannica.com/topic/Occams-razor

[22] LEIBOVICH-RAVEH, Tali, Daniel LEWIS, Saja AL-RUBAIEY, Kadhim AL-RUBAIEY a Daniel ANSARI. A new method for calculating individual subitizing ranges [online]. 2018 [cit. 2022-01-04]. Available from: https://www.researchgate.net/publication/325868989

[23] BONACCORSO, Giuseppe. Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning. Birmingham: Packt Publishing, 2017. ISBN 978-1-78588-962-2.

[24] FRANK A. FARRIS. The Gini Index and Measures of Inequality. The American Mathematical Monthly [online]. 2010, 117(10) [cit. 2022-01-04]. ISSN 00029890. Available from: doi:10.4169/000298910x523344

[25] BROWNLEE, Jason. How To Implement The Decision Tree Algorithm From Scratch In Python. Machine Learning Mastery [online]. 2016 [cit. 2022-01-04]. Available from: https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

[26] YIU, Tony. Understanding Random Forest. Towards Data Science [online]. 2019 [cit. 2022-01-04]. Available from: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[27] ASLAM, Javed A., Raluca A. POPA a Ronald L. RIVEST, MARTINEZ, Ray a David WAGNER, ed. On Estimating the Size and Confidence of a Statistical Audit. Proceedings of the 2007 USENIX/ACCURATE Electronic Voting Technology Workshop [online]. Boston, Massachusetts: USENIX, 2007 [cit. 2022-01-04]. Available from: http://www.usenix.org/events/evt07/tech/full_papers/aslam/aslam.pdf

[28] SRIVASTAVA, Tavish. Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm (with implementation in Python R). Analytics Vidhya [online]. 2014 [cit. 2022-01-04]. Available from: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

[29] UNZUETA, Diego. Convolutional Layers vs Fully Connected Layers. Towards Data Science [online]. 2021 [cit. 2022-01-04]. Available from: https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b

[30] REED, Russell. Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. Cambridge, Massachusetts: The MIT Press, 1999. ISBN 978-0262527019.

[31] BROWNLEE, Jason. Loss and Loss Functions for Training Deep Learning Neural Networks. Machine Learning Mastery [online]. 2019 [cit. 2022-01-04]. Available from: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[32] GODOY, Daniel. Understanding binary cross-entropy / log loss: a visual explanation. Towards Data Science [online]. 2018 [cit. 2022-01-04]. Available from: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[33] Introduction to optimizers [online]. 2018 [cit. 2022-01-04]. Available from: https://algorithmia.com/blog/introduction-to-optimizers

[34] Scrapy: A fast and powerful scraping and web crawling framework [online]. [cit. 2022-01-04]. Available from: https://scrapy.org/

[35] WebDriver. Selenium [online]. [cit. 2022-01-04]. Available from: https://www.selenium.dev/documentation/webdriver/

[36] WILLAERT, Jorrit. How To Calculate the Mean and Standard Deviation — Normalizing Datasets in Pytorch. Towards Data Science [online]. 2021 [cit. 2022-01-04]. Available from: https://towardsdatascience.com/how-to-calculate-the-mean-and-standard-deviation-normalizing-datasets-in-pytorch-704bd7d05f4c