Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Predicting sports matches with neural models

Diploma thesis

Aleksandra Pereverzeva

Master's programme: Open Informatics
Branch of study: Data Science
Supervisor: Ing. Gustav Šír

Prague, January 2021

Thesis Supervisor:
Ing. Gustav Šír
Intelligent Data Analysis
Faculty of Electrical Engineering
Czech Technical University in Prague
Resslova 307/9
120 00 Prague 2
Czech Republic

Declaration

I hereby declare I have written this diploma thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis.

In Prague, January 2021

.....
Aleksandra Pereverzeva

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Pereverzeva Aleksandra** Personal ID number: **453454**

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

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Data Science**

II. Master's thesis details

Master's thesis title in English:

Predicting sports matches with neural models

Master's thesis title in Czech:

Predikce sportovních zápasů s neurálními modely

Guidelines:

Advanced statistical methods are rapidly changing the industry of predictive sport analytics. However, much of the area still relies on the more conservative expert domain knowledge. Similarly, most current predictive sport models rely on manually crafted features for each sport. The purpose of this project is to employ the latest trends from deep learning against SotA approaches and actual market odds from the domain of predictive sports. The main focus of this work is to explore general principles of representation learning (embedding) across variety of sports without any specific features. Student is expected to encode own insights within the model architectures, analyze, optimize and perform sound experimental evaluation against real SotA from the industry.
1) Review related work in using statistical and neural models for predictive sports analytics.
2) Collect a statistically significant amount of match results and market odds from different sports.
3) Perform standard filtering, cleansing, and data exploration.
4) Propose different architectures for neural representation learning of players/teams over different competitions and match structures.
5) Evaluate against SotA, such as selected statistical models and ratings, as well as real market.
6) Assess viability of the representation learning idea within the domain.

Bibliography / sources:

Haghighat, Maral, Hamid Rastegari, and Nasim Nourafza. 'A review of data mining techniques for result prediction in sports.' Advances in Computer Science: an International Journal 2.5 (2013): 7-12.
Bunker, Rory P., and Fadi Thabtah. 'A machine learning framework for sport result prediction.' Applied computing and informatics 15.1 (2019): 27-33.
McCabe, Alan, and Jarrod Trevathan. 'Artificial intelligence in sports prediction.' Fifth International Conference on Information Technology: New Generations (itng 2008). IEEE, 2008.
Hubáček, Ondřej, Gustav Šourek, and Filip Železný. 'Exploiting sports-betting market using machine learning.' International Journal of Forecasting 35.2 (2019): 783-796.
Hubácek, Ondrej, Gustav Šourek, and Filip Zelezný. 'Score-based soccer match outcome modeling–an experimental review.'

Name and workplace of master's thesis supervisor:

Ing. Gustav Šír, Department of Computer Science, FEE

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

Date of master's thesis assignment: **24.06.2020** Deadline for master's thesis submission: **05.01.2021**

Assignment valid until: **19.02.2022**

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

III. Assignment receipt

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

.	
Date of assignment receipt	Student's signature

Abstract

This thesis explores the problem of predicting sports results and offers two approaches that utilize neural networks. The first approach is a traditional artificial neural network with the embedding of the individual teams. The second one is a relatively new approach that employs Convolutional Graph Neural Networks for team representation. The innovation of this work is that the models do not utilize any sport-specific features. Instead, the models are supposed to learn using merely the results of the past matches. The models are created, trained, and tested on two sports domains: soccer and ice hockey. The results turned out to be satisfactory, taking into consideration the generality of the models. However, the resulting models cannot yet compete with state-of-the-art models and market systems.

Keywords: sports prediction, artificial neural networks, graph neural networks, embedding

Abstrakt

Tato práce prozkoumává problém predikce sportovních výsledků a nabízí dva přístupy řešení pomocí neuronových sítí. První přístup je tradiční umělá neuronová síť s embeddingem jednotlivých týmu. Druhé řešení je relativně nový přístup používající Konvoluční Grafové Neuronové Sítě pro reprezentaci týmů. Práce je inovativní tím, že modely nepoužívají vlastnosti specifické pro jednotlivé sporty. Modely se učí na základě výsledků minulých zápasů a jsou vytvořeny, natrénovány a otestovány na dvou doménách: fotbalu a hokeji. Výsledky modelů jsou uspokojivé, pokud se bere v potaz jejich obecnost. Nicméně, výsledné modely ještě nemůžou soupeřit s nejmodernějšími modely a systémy existujícími na trhu.

Klíčová slova: predikce sportovních výsledků, umělé neuronové sítě, grafové neuronové sítě, embedding

Acknowledgements

I thank my supervisor for experienced guidance and thoughtful advice.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

List of Tables

List of Figures

List of Acronyms

ANN artificial neural network. 3, 15

ANOVA Analysis of Variance. 15

ConvGNN Convolutional Graph Neural Network. ix, xii, 7, 8, 22, 24, 28, 29, 31, 33–35, 40

GNN Graph Neural Network. ix, 1, 6, 21–24, 29, 33, 37, 39

LSTM Long Short-Term Memory. 15, 16

NHL National Hockey League. 20

RFC Random Forest Classifier. 16

RPS Rank Probability Score. 10

Contents

Chapter 1

Introduction

Sports betting and sports predictions have accompanied humankind for at least a couple of thousands of years, as the first records of such activity date back to Ancient Greece and Ancient Rome [1]. Over the years, the popularity of the sport categories changed a lot, and so did the legal status of the bets. The sports differed across the countries and continents, the rural and urban areas, changed according to the players' social statuses. Nowadays, even in the age of e-sports, predictive analytics in traditional sports is a multi-million dollar industry. With a recent rise of new machine learning techniques, a new hope was put into resolving the problem of the accurate sports results predictions, however high accuracy solutions for it is still a hot topic and a challenge. One possible way to approach this challenge is to use neural networks that have shown promising results in various other fields.

Those approaches need some input data to describe the teams. These representations can be either explicitly provided by the data, where different features that may affect the outcome of the game are supplied to the model, and it learns to weigh these parameters, combine and transform them in order to output the predictions for the result of the game later; or implicit, where it is assumed that a model learns the representations of the teams themselves and their behaviour during a training phase. This work focuses on the implicit method. It is a relatively new approach, and it has not been actively used for predicting sports results. One way to obtain these representations is by using an embedding layer for each team, further detailed in Section 2.2. Another method is designing some structure that helps to gain insight into relations between the teams and turn it into the teams' representations. That can be achieved using the Graph Neural Networks (GNNs). The idea behind it is that representing the teams as a spatial structure can stimulate the information passing and support a team's knowledge about its strength relative to others while also artificially creating the useful features that will help the model predict the outcome of the match. This will be further discussed in Sections 2.3 and 4.2.2.

1.0.1 Goal

This work aims to test several approaches to representation learning using neural methods for sports analysis. Particularly, it will explore two of such approaches, and train the respective models using the proposed methods on the acquired data. The data will consist of two data sets: soccer and ice-hockey. After that, the work will select the models with the best performance and compare their viability with the state-of-the-art solutions and the market odds.

1.0.2 Problem formulation

The task represents a classification problem with three classes: 'Win' for the home team, 'Lose' for the home team, and 'Draw'. In order to bring the problem closer to real life, the predictions will be made pre-match (as opposed to predicting online, where the predictions of the outcome of the game can be changed up to n minutes before the game ends).

1.0.3 Constraints

The constraints on the work are the following:

- Predictions: the predictions must only use the information available before the game starts.
- Data: features must be extracted implicitly using embedding. Sport-specific features are not used.

Chapter 2

Background theory

The following chapter overviews the theoretical basis necessary to understand the work. The first section reviews the Artificial Neural Networks and technique necessary to train them. The next sections explain techniques of Embedding and Graph Neural Networks. The last section addresses the evaluation metrics.

2.1 Artificial Neural Networks

Artificial neural network (ANN) is an attempt to mimic the mammalian brain in solving problems. The idea behind this is an imitation of the underlying structure of the brain nerve cells (neurons) and connections between them (axons) that send impulses. This idea is implemented into the ANNs as nodes resembling the neurons that can later be organized into layers and node links that operate similarly to the axons. Each node receives some input data, transforms it by performing a simple operation on it (usually summation, multiplication, exponentiation), and then produces the output, which can be used as an input to some other nodes. ANN is a learning algorithm that can lead to better performance in the tasks where the traditional approaches stall, especially in classification, image and text recognition, and others. When implemented, neurons are represented as a non-linear function:

$$f(X) = \sigma(W^T X + b),$$

where W represents the neuron's weights, b — bias, and σ is the activation function that adds the non-linearity.

2.1.1 Backpropagation

Backpropagation is an essential algorithm for updating the neurons' weights that provides the actual learning of the network. First, the model computes a result based on its weights, and it is called a forward pass. For measuring how far the prediction of the network is from the actual

value, the loss function is used. [1] After that, a backward pass is ready to be performed for every node from the network's end to its beginning. It recursively applies the chain rule to calculate the gradients of the neural network weights with respect to a loss function. More specifically, two messages are computed for every neuron: a partial derivative of the neuron's output with respect to its input and a partial derivative of the neuron's output with respect to the neuron's weights. During the backward pass, the gradient with respect to each of the inputs is multiplied with the gradient that came back from the output and then passed to the previous neuron. The gradient with respect to the neuron's weights are as well multiplied by the gradient returned from the next layer, and the result can be later used to update the weights of the neuron by using the gradient methods to find the weights that minimize the loss, for example, gradient descent with the update function at iteration $t + 1$:

$$\theta^{t+1} = \theta^t - \eta^t \nabla \mathcal{L}(\theta^t), \tag{2.1}$$

where θ represents the parameters of the network (weight matrices), η — the learning rate and $\nabla \mathcal{L}(\theta)$ is the gradient of the loss function [2]–[4]. Adam is another widely optimization algorithm that utilizes adaptive learning rate and momentum for a more effective optimization process [5].

2.1.2 Dropout

Dropout is a technique to regularize the network and provide a computationally cheap way to reduce data overfitting. Dropout essentially helps to simulate network ensembles known to reduce data overfitting, which, however, in their natural form require a higher computational cost. This technique operates by randomly dropping some nodes with a predefined probability during training. During validation and testing, all the nodes and weights associated with them are present; weights are multiplied by the probability of the node being present during training. The network of n neurons with applied dropout can be viewed as a combination of 2^n possible reduced architectures. However, all the weights are still shared across them. That is why the total number of parameters stays within the limit of $O(n^2)$ in the case of a fully connected network [6]. An example of a dropout can be viewed in Figure 2.1.

2.1.3 Cross-entropy loss

A softmax classifier is a multinomial interpretation of the binary logistic regression. For a class k the computation of the softmax function takes place as follows:

$$\sigma_k(s) = \frac{e^{s_k}}{\sum_j^N e^{s_j}}, \tag{2.2}$$

[1] The loss function should satisfy two assumptions:

1. The loss function can be written as an average over all examples: $\mathcal{L} = \frac{1}{n} \sum_i^n \mathcal{L}_i$.
2. The loss function should be a function of the neural network's outputs: $\mathcal{L} = \mathcal{L}(\hat{y})$

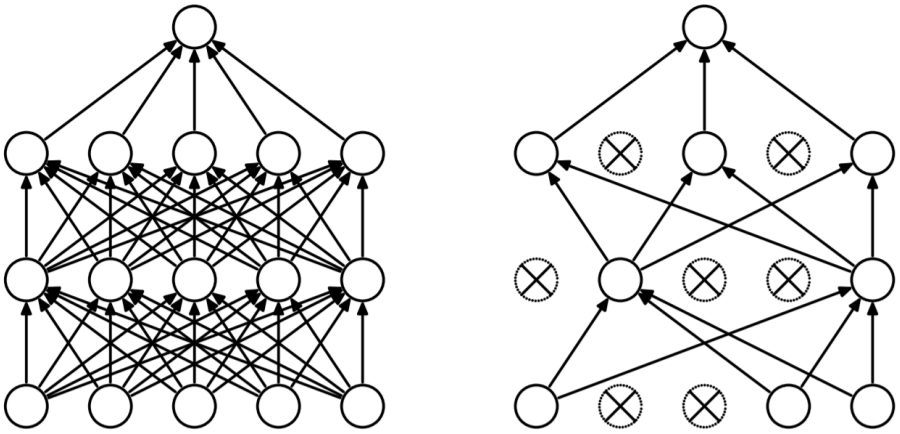


Figure 2.1: Example of dropout applied to a neural network. A graph on the left represents a neural network with two hidden layers. A graph on the right shows what a network on the left could look like after applying a dropout to it: the crossed-out neurons and all edges incident to them are dropped. Image taken from [6].

where s is the vector from a scoring function, for example, a neural network, and N is the number of classes [2]. The output of the softmax function defines the probability of an observation belonging to each class. The multinomial cross-entropy loss is then acquired by plugging the output from the softmax function into the negative log likelihood loss:

$$\textit{Cross Entropy Loss}(s) = \sum_{k}^{N} y_k log(\sigma_k(s)), \tag{2.3}$$

where the y_k is the binary indicator if k is the correct classification for the observation s [7].

2.2 Embedding

Embedding is a technique that allows transforming the data into an implicit structure that behaves in the same way as an explicit ranking system [2]: it assigns each input record a high-dimensional vector and, during training, transforms those vectors so that the records with a similar meaning get assigned close vectors. Although initially used to meaningfully vectorize words [8], in the case of one-dimensional output, the output can be interpreted as the relative power of the record. Embedding can be further defined as a generalization of the rating systems to n dimensions since each dimension of the resulting vector can represent itself a rating. Another problem embedding can deal with is helping with sparse matrices by reducing the dimensions of very high-dimensional data to a sensible amount of dimensions that still reflects all the underlying relations [9]. An example of word embedding can be found in Figure 2.2.

[2] A ranking system is usually understood as a system that can order input items in a meaningful way by comparing them. On the other hand, a rating system assigns a score to those input items based on some standard scale and does not necessarily compare them. However, in the literature related to sports analytics, these concepts are used interchangeably because the sports analytics systems both assign a score value to the teams or players and order them based on that score.

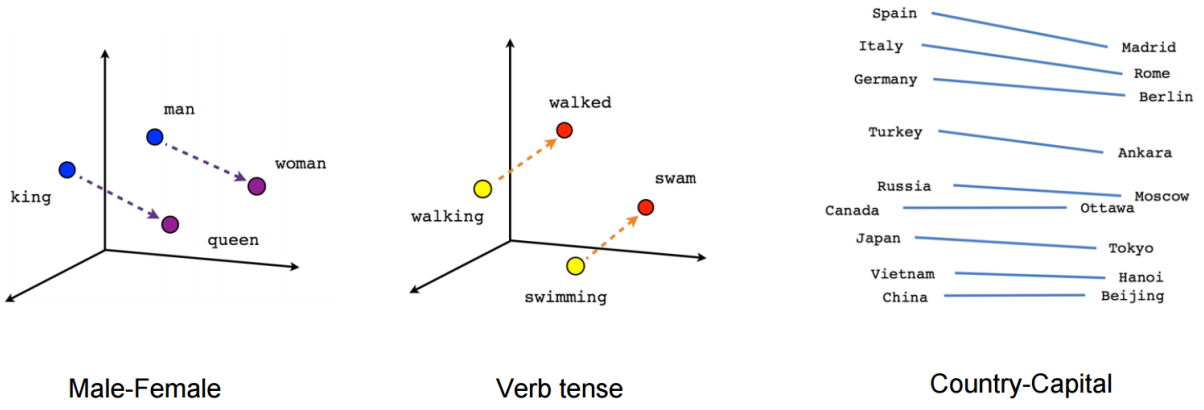


Figure 2.2: Example of the word embedding. The main idea is that the objects that are close to each other in natural meaning stay close to each other in embedding projection. Image taken from [10].

The parameters of the embedding can (and actually need to) be learned using the same method as used in regards to other layers – backpropagation. The vectors for each sample in the data set are generated containing small random non-zero values and then updated during the training phase.

2.3 Graph Neural Networks

GNN is a type of neural network architechture that utilizes the underlying structure of the data. GNNs can be used for multiple purposes [11]:

- Node-level: node classification usually conducted in a semi-supervised manner where the nerwork predicts the affiliation of the unlabeled node with a certain class based on labeled ones. This type of task is often effectively solved by stacks of Graph Convolutional Layers.
- Graph-level: predicting the label for the entire graph. This problem is tackled by combinations of graph convolutional layers, graph pooling layers, and readout layers.
- Edge-level: edge classification, link prediction.

Graphs can be used many problems from real life since they provide important connections (represented as edges) between individual items (represented as nodes).
A graph G with a total number of nodes n can be defined as a pair of two sets: a set of vertices V and a set of edges E. Neighbors of a given vertex v constitute a subset of nodes: $N(v) = \{u \in V | (v, u) \in E\}$. Each vertex can have an ordered set of attributes in a form of a vector x_v.
Wu, Pan, Chen, *et al.* [11] proposes the following taxonomy of GNNs:

- Recurrent Graph Neural Networks;
- Convolutional Graph Neural Networks;

- Graph autoencoders;
- Spatial-temporal Graph Neural Networks.

In this work the main focus will be addressed onto the Convolutional Graph Neural Networks (ConvGNNs), which generalize the process of convolution from grid data (such as images) to graph structures. The key idea behind ConvGNN is generation of the node's representation based on the node's attributes x_v and node's neighbors' attributes $x_u, u \in N(v)$. Multiple Graph Convolutional layers can be stacked to achieve higher quality representation of the nodes. In that case each node is constructing its representation from further neighborhoods. An example of such network can be found in Figure 2.3 Wu, Pan, Chen, *et al.* [11] distinguishes between two

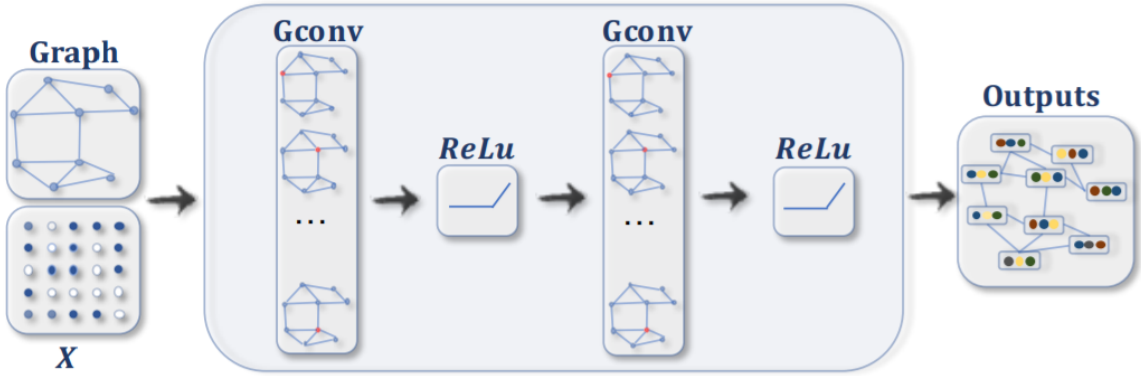


Figure 2.3: Example of the ConvGNN and how it can be used for node classification. Input is a graph structure and an attribute (feature) matrix. Each graph convolutional layer of the ConvGNN (depicted as Gconv) provides feature aggregation, the output of which then undergoes a non-linear transformation by an activation function (depicted as ReLu). The learned representations of the nodes can then be used to their classification. Image taken from [11].

streams (types) of convolutional graph neural networks:

1. Spectral-based ConvGNN;
2. Spatial-based ConvGNN.

The foundation for this type of ConvGNNs comes from graph signal processing. Spectral-based ConvGNNs assume the underlying graphs in the data to be undirected. The mathematical representation them is based on the normalized graph Laplacian matrix, that can be defined as the follows:

$$L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}, \tag{2.4}$$

where D is a diagonal matrix of node degrees and A is an adjacency matrix for the graph. Since one of the properties of the normalized graph Laplacian matrix is it being symmetric positive semidefinite, the matrix can be further factored as:

$$L = U \Lambda U^T, \tag{2.5}$$

where $U \in R^{n \times n}$ is the matrix of eigenvectors, Λ is a matrix of eigenvalues. Here graph signal processing offers a concept of filters - a graph convolutional operation that removes noises from graph signals. A filter g_θ can be defined as:

$$g_\theta = diag(U^T g), \tag{2.6}$$

where $g \in R^n$ is a vector of Fourier coefficients. Then the filter can be applied to an input signal x as follows:

$$g_\theta \star x = U g_\theta U^T x. \tag{2.7}$$

All the graph convolutional neural networks follow this definition and are distinguished by the preference of the filter g_θ.

Spatial-based ConvGNN is a method that applies convolution based on the node's spatial relations. Its key idea is the direct propagation of the information from a node to its neighbors using the connections.

Graph Convolutional Layer: GraphConv

Morris, Ritzert, Fey, *et al.* [12] propose a new graph neural network operator as follows:

$$\mathbf{x}'_i = \mathbf{\Theta}_1 \mathbf{x}_i + \mathbf{\Theta}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j, \tag{2.8}$$

where $\mathbf{\Theta}_1, \mathbf{\Theta}_2 \in R^{k \times f}$ define trainable parameters (k is a hyperparameter) different for a node and for its neighbors, e_{ij} denotes weight of an edge from i to j and $x_i \in R^f$ denotes the representation of a node i before its update, f is a number of features or, in other words, depth of representation. The summation in terms of means of aggregation is optional - it can be changed to the mean or max functions.
GraphConv is a spatial-based ConvGNN.

Graph Convolutional Layer: ChebConv

Defferrard, Bresson, and Vandergheynst [13] describe a spectral-based ConvGNN layer. The filter is approximated by Chebyshev polynomials of the first kind of degree K.
The Chebyshev polynomials are obtained from a recursive relation as follows [14]:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n_1}(x). \end{aligned} \tag{2.9}$$

The filter is then approximated and applied to the feature vector using a scaled and normalized

Laplacian matrix. The new representation feature vector of a node can be defined as [15]:

$$\mathbf{X}' = \sum_{k=1}^{K} \mathbf{Z}^{(k)} \cdot \mathbf{\Theta}^{(k)}, \tag{2.10}$$

where Θ is the matrix of trainable parameters, $\mathbf{Z}^{(k)}$ is the approximated filter based on the Chebyshev polynomials (where the polynomials order and filter size K is a hyperparameter) and is computed recursively according to the following:

$$\begin{aligned} \mathbf{Z}^{(1)} &= \mathbf{X} \\ \mathbf{Z}^{(2)} &= \hat{\mathbf{L}} \cdot \mathbf{X} \\ \mathbf{Z}^{(k)} &= 2 \cdot \hat{\mathbf{L}} \cdot \mathbf{Z}^{(k-1)} - \mathbf{Z}^{(k-2)}, \end{aligned} \tag{2.11}$$

where $\hat{\mathbf{L}}$ is the modified Laplacian scaled by the largest eigenvalue of the initial $\mathbf{L}$:

$$\hat{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I} \tag{2.12}$$

Thanks to the fact that the computed filters are localized in space, they are computed efficiently, without the need to compute the eigenvalues and are universal to the graph structure, meaning that the filters performance does not depend on the graph size or internal structure.

Graph Convolutional Layer: GCNConv

Kipf and Welling [16] propose a new graph convolutional layer that is a special case of ChebConv described in Section 2.3. The filter size is fixed to $K = 1$ and the largest eigenvalue for the Laplacian matrix normalization is approximated to be $\lambda_{\max} = 2$. Under this approximation the application of the filter can be simplified to:

$$g_{\theta'} \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x. \tag{2.13}$$

The expression can be further normalized by reducing it to the following state:

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-1/2} \mathbf{X} \mathbf{\Theta}, \tag{2.14}$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is an adjacency matrix with self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ denotes its degree matrix and $\mathbf{\Theta}$ is a trainable parameter matrix. The computation can be improved by computing the $\hat{\mathbf{A}}$ as $\mathbf{A} + 2\mathbf{I}$ [17].

Kipf and Welling [16] state that this renormalized propagation model improves both efficiency and prediction capabilities, however, one can notice that the number of parameters is drastically lower and a lower number of degrees of freedom can negatively affect generalization.

2.4 Evaluation

2.4.1 Accuracy

Accuracy is the most commonly used metric for classification tasks. It is a ratio of the correctly classified instances to the number of all instances.

2.4.2 RPS

One of the metrics used in the 2017 Soccer Prediction Challenge [18] was Rank Probability Score (RPS). In this work we also adopted it to be able to compare the results achieved. RPS was introduced in 1969 in Epstein [19] as a scoring system for prediction of ordinal categorical variables. Later Constantinou and Fenton [20] utilized it for measuring the model performance on predicting the result of sports events and showed that it is more robust than Brier score and Geometric score since it measures the difference between cumulative distributions of the model's predictions and actual observations.
RPS is calculated according to the following formula:

$$RPS = \frac{1}{r-1} \sum_{i=1}^{r} \left(\sum_{j=1}^{i} p_j - \sum_{j=1}^{i} o_j \right)^2 , \tag{2.15}$$

where the p_j and o_j stand for the predicted probability and actual observation of an instance belonging to the class j, respectively, the r is the number of classes.
This way the RPS measures not only how correct the model was, but also how sure was the model in its predictions.

Chapter 3

Research into related work

The following chapter discusses several existing approaches that were considered before designing the models of this thesis. The models in the following chapter (unless specified differently) were chosen because they use the same type of data – historical match results – for training and testing. In the first section of this chapter are discussed statistical models and ratings, then the focus will turn to the advanced approaches that use machine learning techniques. The last section will review an approach that uses explicit features using the example of the combined RPI [21].

3.1 Statistical models and ratings

This section is dedicated to the models that either explicitly assume that the data comes from some probability distribution or try to learn some implicit distribution using various methods discussed below. They try to determine how good a team is compared to other teams and give it a relative score. The prediction for a sports match between two rival teams then can be made by comparing the scores for those two teams.

3.1.1 The Poisson models

The Poisson models were proposed in 1982 in the article [22]. The paper suggested that the number of goals scored by each team corresponds to the independent variable from the Poisson distribution. Following that assumption, the probability of a variable that corresponds to the number of goals scored by the home team (G_H) being x and the respective variable for the away team (G_A) being y is given by

$$P\left(G_H = x, G_A = y | \lambda_H, \lambda_A\right) = \frac{\lambda_H^x \cdot e^{-\lambda_H}}{x!} \cdot \frac{\lambda_A^y \cdot e^{-\lambda_A}}{y!}, \tag{3.1}$$

where λ_H and λ_A represent the means of the Poisson distribution for the home team and the away team respectively, and can be interpreted as the teams' scoring rates. The model can be

modified to include multiple parameters for each team such as attack (Att) and defense (Def) strengths as well field advantage(Adv), which then can be used to calculate the scoring rates:

$$\begin{aligned} \lambda_H &= e^{Att_H - Def_A + Adv_H} \\ \lambda_A &= e^{Att_A - Def_H}. \end{aligned} \tag{3.2}$$

A recent paper of Ley, Wiele, and Eetvelde [23] argues that the number of parameters can be effectively reduced using the modified parameters – instead of two parameters that represent strengths of the team, one complex strength (Str) can be introduced:

$$\begin{aligned} \lambda_H &= e^{Str_H - Str_A + H} \\ \lambda_A &= e^{Str_A - Str_H}. \end{aligned} \tag{3.3}$$

There are multiple recent papers [23], [24] that prove the competitiveness of the Poisson model among other ranking models – both papers concluded that the Poisson model outperformed other models in the conducted tests.

3.1.2 Elo ratings

Elo rating system is one of the most used one in sports and games since its implementation by USCF [1] in 1960 [25], . The idea is to quantify the relative skill level of the individual players [26]. It is done according to this formula [27]:

$$\begin{aligned} E_H &= \frac{1}{1 + c^{(R_A - R_H)/d}} \\ E_A &= 1 - E_H \end{aligned} \tag{3.4}$$

Then the prediction of the result of the match is just simply comparing the expected skill levels of the individual players:

$$E_{S_H} = \begin{cases} \text{home team win} & \text{given } E_H > E_A \\ \text{a draw} & \text{given } E_H = E_A \\ \text{away team win} & \text{given } E_H < E_A \end{cases} \tag{3.5}$$

The ratings of the players are then updated according to this formula:

$$S_H = \begin{cases} 1 & \text{given a home team win} \\ 0.5 & \text{given a draw} \\ 0 & \text{given away team win} \end{cases} \tag{3.6}$$

[1] United States Chess Federation

$$\begin{aligned} R_H^{t+1} &= R_H^t + k(1+\delta)^\gamma \cdot (S_H - E_H) \\ R_A^{t+1} &= R_A^t - k(1+\delta)^\gamma \cdot (S_H - E_H) \end{aligned} \tag{3.7}$$

where

- R_H and R_A stand for rating of home teams and away teams respectively;
- c and d are metaparameters;
- γ is a metaparameters that defines the impact of goal difference on rating update;
- E_H and E_A are the the expected outcomes for the home and away team respectively;
- E_{S_H} is the expected outcome of the match for the home team;
- S_H is the real result of the match for the home team;
- t stands for the order number of the match;
- δ represents an absolute value of the goal difference between the home team and the away team;
- k is the learning rate.

This kind of rating system is simple and proved to be working in traditional sports, such as chess and football, as well as e-sports, such as Age of Empires [28].

3.1.3 pi-ratings

Constantinou and Fenton [29] suggest a simpler rating system specifically for soccer, but which is applicable for any sport where the score is the main indicator of teams strength. The pi-ratings outperformed Elo rating system, showed to be competitive with the bookmaker's predictions and won the 32.81% of the placed bets and in the end made profit of 15.4% of the initial bet. In pi-rating system, the average score remains 0 and cannot inflate over time. Each team's rating consists of two sub-ratings: home and away; the overall team's rating is just an average of those two sub-ratings. The mentioned home (R_{aH}) and away (R_{aA}) sub-ratings of a team a are then updated respectively as follows:

$$\begin{aligned} \hat{R_{aH}} &= R_{aH} + \lambda\psi_H(e) \\ \hat{R_{aA}} &= R_{aA} + \upsilon(\hat{R_{aH}} - R_{aH}), \end{aligned} \tag{3.8}$$

where λ, υ are the learning rates, $\psi(e)$ is the fuction that ensures the win has more effect on the rating that the goal difference e.

3.1.4 PageRank

The RageRank ratings origin from an algorithm devised for ranking pages in Google Search engine results [30]. The idea behind it is measuring the importance of a website page by counting the number and quality of links leading to it. That leads to the assumption that the more important is a website, the higher number of other websites refer to it.
However, the algorithm can be generalized to be applied to the area of sports ratings [31]. A series of competitions can be represented as a graph, where the vertices represent the n teams and the edges represent the matches conducted between the pairs of the individual teams. The hyperlink (adjacency) matrix H element standing for all the matches played between teams i and j can be defined as follows [24], [32]:

$$H_{ij} = \frac{\sum_m w(m) S_j(m)}{\sum_m w(m)}, \tag{3.9}$$

where $S_j(m)$ is the number of points of the team j scored against the team i at match m, $w(m)$ is the weighing of the match m. From that hyperlink matrix rating of each team can be computed as described in Govan, Meyer, and Albright [31].

3.1.5 Berrar ratings

Berrar ratings suggest predicting not only the ternary outcome of the match but also the goals scored by each team [33]. As opposed to Elo, Berrar ratings use offensive and defensive strengths to rank the team and not the rating itself. Expected number of scored goals for the home and away teams are calculated according to the following formula [33]:

$$\begin{aligned} \hat{G}_H &= \frac{\alpha}{1 + e^{-\beta_H(o_H - d_A) - \gamma_H}} \\ \hat{G}_A &= \frac{\alpha}{1 + e^{-\beta_H(o_A - d_H) - \gamma_A}} \end{aligned} \tag{3.10}$$

Then the offensive and defensive strengths of the home team are updated as follows:

$$\begin{aligned} o_H &+= \omega_{o_H}(G_H - \hat{G}_H) \\ d_H &+= \omega_{d_H}(G_A - \hat{G}_A) \end{aligned} \tag{3.11}$$

The updates for the away team are conducted in the similar fashion.

- $\hat{G}_H$ and $\hat{G}_A$ stand for the expected goal scores for the home team and away team respectively;
- G_H and G_A are the actual goal scores for the home team and away team respectively;
- α defies the maximum possible number of goals a team can score
- β and γ are the metaparameters that can be interpreted as slope of the logistic function and the bias respectively, metaparameters are individual for the home and away team as

opposed to Elo;

- ω stands for the learning rate for the strengths update.

3.1.6 TVC: Time Varying Coefficients

TVC assume that the points scored throughout the season have different weight on the strength of the team at the end of the season [34]. To translate this effect into the model, the TVC allows some coefficients to vary with time. The strength of the team i at time t can be calculated as follows:

$$\lambda_{it} = \sum_{k \in V} \gamma_k(m_{it}) x_{itk} + \sum_{k \notin V} \beta_k x_{itk}, \tag{3.12}$$

where x_{it} is the feature vector associated with the team i at time t, V is the set of coefficients features that can vary in time and the m_{it} is the number of matches played by i by the time t. Tsokos, Narayanan, Kosmidis, *et al.* [34] later embedded this team's strength into the Bradley-Terry model to generate predictions:

$$p(y_{ijt} = 1) = \frac{exp(\lambda_{it})}{exp(\lambda_{it}) + exp(\lambda_{jt})}, \tag{3.13}$$

where y_{ijt} is 1 when team i beats j at time t. This model took the third place of the 2017 Soccer Prediction Challenge.

3.2 Advanced models

The following chapter reviews some of the resources that used advanced computational techniques to predict sports results.

Arabzad, Tayebi Araghi, Sadi-Nezhad, *et al.* [35] proposed an earlier attempt to predict the sports results based on the historical data using ANN. The paper tried to predict the outcome of 8 matches. To achieve robustness of the model, the predictions were generated 30 times. The model consisted of 2 hidden layers with 20 neurons in each of them. The 10 input features included recent average points and all time average points for each team. The goal was to predict the actual score of the game. The output was then analysed using graphical and statistical analysis (ANOVA). The result shows that score of 1 of 8 matches was predicted correctly and the victory team of 5 of 8 matches was predicted correctly.

The article of Nyquist and Pettersson [36] proposes Long Short-Term Memory (LSTM) architectures of recurrent neural networks and reports the accuracy of up to 98.63%. However, this high accuracy is achieved thanks to training the models to predict the next 15 minutes of the game, meaning that the models predict the match's outcome with the information about the whole match minus 15 minutes. This way of evaluation was chosen due to the authors' goal to predict the result at any given moment and not just at the end of the game. The best

accuracy of predictions of the outcome of the matches before their start is close to 44%. It is worth mentioning that this work uses embedding to create input for the network. However, the embedding used in this paper creates the vector based not only on the team name but also on the soccer-specific features, such as penalties, card types, goal types, lineup, and substitution. After the features are embedded, they serve as the input to the recurrent neural network with LSTM units. The network is then complemented with the softmax classifier to produce the predictions for each class. The fact that the paper's authors had access to the data sampled relatively densely appeals to the idea of modeling their predictor using recurrent neural networks with LSTM units that work best with sequences and can foresee subtle changes throughout the timeline.

Pugsee and Pattawong [37] used a Random Forest Classifier (RFC) for predicting the soccer match results. RFC is a machine learning technique that involves training multiple classifiers (decision trees) on randomized training data and then combining their results to obtain the prediction. Each decision tree's attributes are selected based on the information gain, representing each attribute's knowledge value. A set of the selected attributes then divides the data and produces a prediction for one tree. Combining several decision trees into random forests helps to tackle the problem of decision tree overfitting the data. The data used to train the RFC consisted of 18 soccer-specific features from the English Premier League. The accuracy of predictions on the testing data by the proposed model is 80%.

Hubáček, Šourek, and Železný [38] proposed an approach for prediction of sport matches outcomes using gradient boosted trees. The proposed model utilized multiple latent features extracted from the data, such as pi-ratings and PageRank rating, multiple features representing current and historical strengths when played at home and away venues, features that relate to leagues. The work discussed in this paper used Xgboost [39] implementation of the gradient boosted trees. As a result, the pi-ratings showed to be the most influential features, while the features associated with the teams' current form were used the least. The paper also proposed another model where the features were ground facts that defined the relations between teams: a historical result of the given game between two teams being a home victory, draw or away victory, matches being played in a league, a team scoring more than a given number of goals in a given match, and a team belonging to a specific rating group when playing a given match. The constructed ground facts provided input for the RDN-Boost algorithm, which is an instance of a relational dependency network function approximators [40]. The Xgboost model got the best RPS on the validation and testing data and is the winning model of the 2017 Soccer Prediction Challenge (the accuracy on testing data is 52.43% and average RPS of 0.2063). However, the relational model performed well as well (average RPS of 0.216).

Constantinou [41] designs a model as a mixture of two models: dynamic ratings and Hybrid Bayesian Networks. The key idea of Dolores (the proposed model) is predicting the outcome

of the match between two teams based on the historical data from other (seemingly unrelated) teams (from other countries, leagues). To accomplish this some more artificially designed data instances were added for matches with identical rating difference (home team score - away team score). This approach allowed to tackle several problems at once: temporality of the data (newer data have higher value for predictions), absence of data for new teams, and predictions for one league based on a different league. The Hybrid Bayesian Network consists of one observable node that describes Rating Discrepancy and 4 hidden nodes: Ability Difference, Goals Scored by Home team, Goals scored by Away team and Predictions. Ability difference can acquire 42 distinct rank values, Rating discrepancy represents a mixture of 42 Gaussian distributions. The example of how Dolores works is described in Constantinou [41]. The model took the second place in the 2017 Soccer Prediction Challenge with the average RPS of 0.2083 and accuracy of 51.46%.

3.3 The combined RPI

This section will shortly discuss an approach that uses sport-specific features for sports predictions.
Combined RPI [21] proposes a new rating approach that incorporates 4 statistics applied to American football matches:

1. RPI;
2. Pythagorean wins;
3. Offensive strategy;
4. Turnover differential.

RPI suggests a score based on the frequency of wins and integrate the score of opponents with some weights.

The Pythagorean wins construct the score based on how many points are scored or surrendered. Offensive strategy projects which way the team chooses to move the ball: by passing or by rushing. The turnover differential is the difference between the takeaways and giveaways for each team, where takeaways are calculated as a sum of interceptions and fumbles that were recovered, while giveaways are the sum of lost interceptions and lost fumbles. These four statistics are normalized (each statistic with individual range) and summed up and produce the output score. The predicted winner is the team with the highest score.

One can notice that three of the four statistics used to calculate combined RPI use data specific to American football. This fact explains why this statistic will not be further considered despite exceptionally high accuracy in predictions (reported accuracy up to 97.14%). Another concern related to the combined RPI is that the reported accuracy suggests that the model overfits the testing data.

Chapter 4

Implementation, Experiments, Evaluation

This chapter is dedicated to the practical part of the work. It reviews the collected data, proposes the solutions of the stated problem, discusses evaluation frameworks and the evaluation process itself and the direct implementation.

4.1 Data

The data was searched on free sources on the Internet. According to the assignment, it is necessary to create models for more than one sport. One obvious candidate was soccer since it is one of the most popular sports in the Czech Republic, as well as globally [42]; besides, it is most commonly reviewed in the relevant works discussed earlier in part 3. Another sport highly popular in the Czech Republic is ice hockey, which also was proved to be a source of identification for Czechs as a nation [43].

4.1.1 Soccer data set

Soccer overview

A professional soccer match involves two teams of 11 players competing against each other in a game during 2 periods of 45 minutes.
Matches are chronologically arranged into seasons. Teams are divided into leagues depending on their professionalism and countries of origin. The structure of leagues is different for each country; usually the leagues consist of 16 to 20 teams.
According to the rules, the teams are awarded with three points for a game ended with a victory, no points for a loss and both teams receive one point if the game ended with a draw.

Season	League	Date	Home Team	Away Team	Home Score	Away Score	Score Difference	Result	Country
2000	GER1	2000-09-16	Bochum	Wolfsburg	2	1	1	W	Germany
2003	ITA1	2004-03-14	Lazio	Udinese	2	2	0	D	Italy
2008	SPA2	2009-02-28	Alaves	Real Sociedad	2	1	1	W	Spain
2008	GRE1	2008-09-28	PAOK	Aris	1	0	1	W	Greece
2014	FRA1	2014-11-22	Monaco	Caen	2	2	0	D	France

Table 4.1: Example of the data from the soccer data set.

Soccer data set overview

The soccer data set is acquired from the Open International Soccer Database for machine learning [44]. It consists of the records from 216743 league soccer matches played in across 52 leagues in 35 countries during the years 2000-2016. The database was specifically produced for the 2017 Soccer Prediction Challenge [18]. From the Figure 4.1 one can notice that the distribution between the results is not even for most leagues and there is a slight bias towards the home victories (40% Home wins, 35% Away wins and 25% Draw). An example of the soccer data

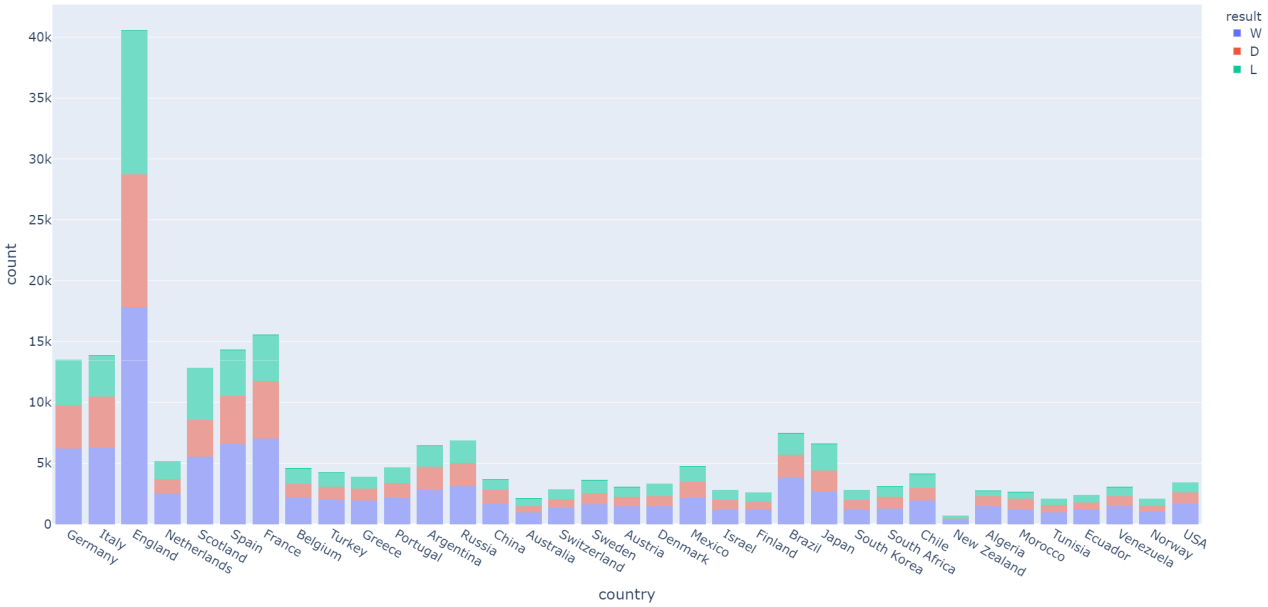


Figure 4.1: Distribution of matches by country and by result: home victory (W), draw (D) and away victory (A)

set can be seen in Table 4.1. Figure 4.2 shows that the number of leagues varies significantly from country to country: while England has five leagues, China only has one. When comparing to Figure 4.1, one can notice that the number of leagues does not necessarily mean a higher number of games for a country; for example, Scotland has four leagues, but the number of played matches is lower than the one of Italy that only has 2. That can be explained by the fact that the data set covers 17 years, and during that time number of leagues can change (for example, for lack of funding); besides that, the number of teams in leagues varies for different countries.

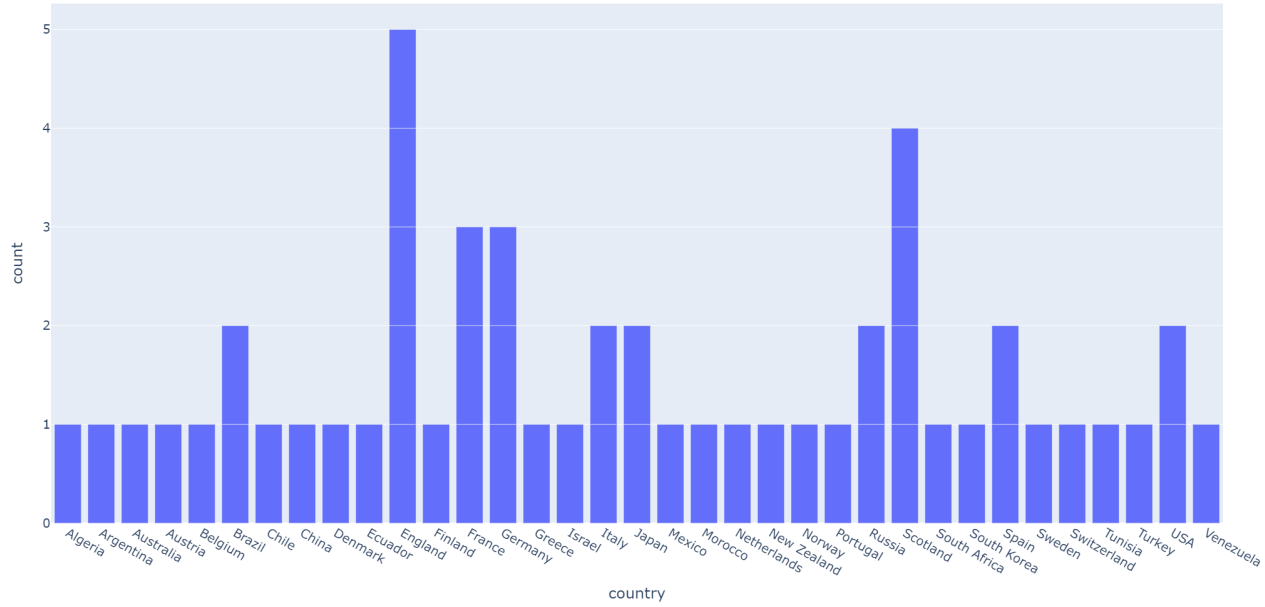


Figure 4.2: Distribution of leagues by country

Another high quality data set, containing the matches results as well as significant number of features and the betting odds, was found at [45], but it will not be further used.

4.1.2 Hockey data set

The hockey data set was acquired by web crawling a web page [46] and then formatted to the same data structure format as the soccer data set.

Hockey overview

The ice hockey is a winter sports game played between two teams of six players each during 3 periods of 20 minutes. The goal of the game is to shoot a puck into the opponent team's net using stick while skating on an ice rink.

National Hockey League (NHL) is a professional ice hockey league that consists currently of 31 teams from United States of America and Canada. Each game plays 82 games during a season: half at a home field and half as a visiting team. The team ranking is based on a point system – two points are awarded for a victory, one point for losing during overtime or in a so called shootout [1] and zero points if the team loses in a regular period.

Hockey data set overview

The table 4.2 shows an example of the data from the hockey data set. The hockey data set

[1] Both shootout and overtime are measures to solve a draw between the teams. If after an additional period (overtime) a winner-determining result is not achieved, a shootout is appointed – the teams representatives shoot the puck until one team prevails.

Season	League	Date	Team home	Team away	Goals home	Goals away	Goal Difference	Result	Country League
1998	NHL	08-10-98	Calgary Flames	San Jose Sharks	3	3	0	D	NHL
1998	NHL	08-10-98	Florida Panthers	Tampa Bay Lightning	4	1	3	W	NHL
1998	NHL	08-10-98	New York Rangers	Philadelphia Flyers	0	1	-1	L	NHL
1998	NHL	09-10-98	Boston Bruins	St. Louis Blues	3	3	0	D	NHL
1998	NHL	09-10-98	Carolina Hurricanes	Tampa Bay Lightning	4	4	0	D	NHL

Table 4.2: An example of the data from the hockey data set.

contains 23968 instances of the sports games of the NHL between the seasons 1998-2018. The number of teams across all those years is only 33 while the current number of teams in the league is 31. This might implicate the strong competitiveness and the fact that there will not be any obvious stronger and weaker teams.

The distribution between the three classes of the results is even less equal for the hockey data set when compared to the soccer data set: Home win 45%, Away win 35%, Draw 20%.

4.2 Proposition of the solution

The purpose of this work is to learn the representations of the teams using embedding. The idea behind it is that the models learn using artificially created features that are adjusted during the training phase. By learning the representations of the individual teams, the model is supposed to simulate the rating systems' behavior. The main advantage of such an approach is the domain independence of the models and that it can be used for any set of events with two adversaries and can later be extended to a higher number of teams or players. Similarly to the simple rating systems, the models use only the historical data of match results. As opposed to the rating systems, there are no hardcoded rules in the explored models, and the models have to simulate the rules by adjusting their parameters during the training phase.

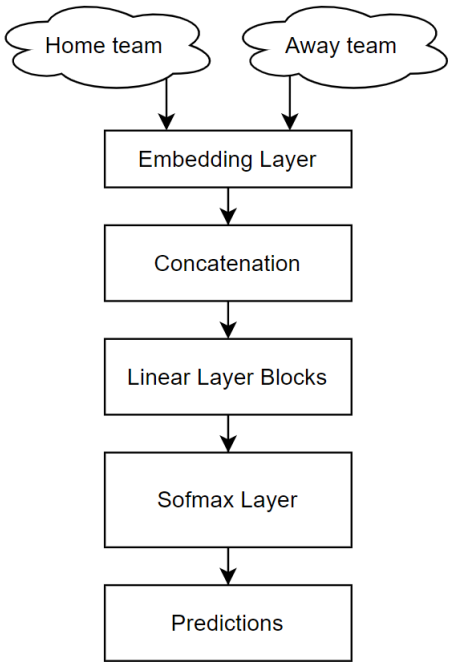
In this work, the following models are proposed and experimented with:

1. Simple ANN with embedding;
2. Graph Neural Network.

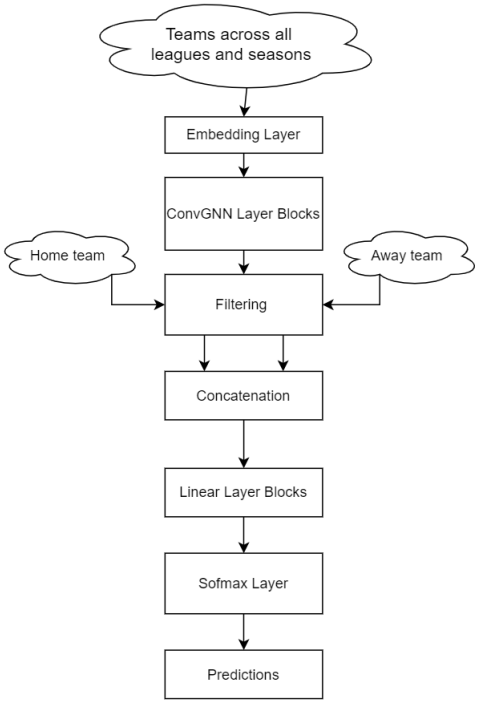
The simple ANN model will learn the representation using only an embedding layer. The GNN will try to improve the embedding learned by the previous model using the additional information about relations between the teams compiled into a graph structure.

4.2.1 Simple ANN with embedding

A simple classification model can be obtained using multiple Linear (fully-connected) layers that are closed by a softmax layer to generate predictions. A diagram of such model is shown in Figure 4.3a. A Linear Layer block represents a combination of a Linear (fully-connected) Layer



(a) A diagram of the Simple ANN with embedding model.



(b) A diagram of the GNN model.

Figure 4.3: Diagrams of the two models.

with an activation function and a possible dropout layer. The representation of all the teams is kept in a single embedding layer.

4.2.2 Graph Neural Network

Once the prediction model is improved, the next most reasonable step is improving the representation of the teams. As the PageRank ratings discussed in Section 3.1.4, graph structure can help to learn the strengths of the individual teams. However, it can be beneficial to learn the representation using the graph structure in higher dimensions. That can be fulfilled by employing the GNN. At first, the match prediction problem in terms of GNN can be recognized as a problem of edge labeling since one graph representation of the historical match data can be described as teams being nodes and matches being connections between two teams. However, that problem can be reformulated into two successive subproblems:

1. Learning teams representation using a graph structure;
2. Learning to predict a result of the match based on those representations.

One can notice that the first subproblem can be interpreted as node classification, the task that was proved to be efficiently solved by ConvGNN [47], [48]. The second subproblem was already tackled by a previous model in Section 4.2.1.

Another problem that needed to be solved was the difference of influence of the game on teams' representations depending on the time component. It is obvious that the victory that happened

last season affects the team's relative strength in the season's finals with a different intensity than a loss that occurred right before the decisive game. That was achieved by weighing the edges of the graph differently depending on the recency of the last game that played between the two teams in question. That is why when the information is propagated from a team's neighbors to the team and from the team to its neighbors, the effect of the information transferred over the edge from node i to node j at time t on the representation is scaled based on last game's recency by multiplying the propagated data with $w^t_{e_{ij}} \in (0, 1) \subset R$. The edge weight can be computed by a linear or an exponential function. Computation by a linear function:

$$w^t_{e_{ij}} = 1 - \frac{t - t_prev}{N}, \tag{4.1}$$

where the t_prev is the time of the previous sport meeting between the two teams and N is the number of a total number of games. Computation by an exponential function:

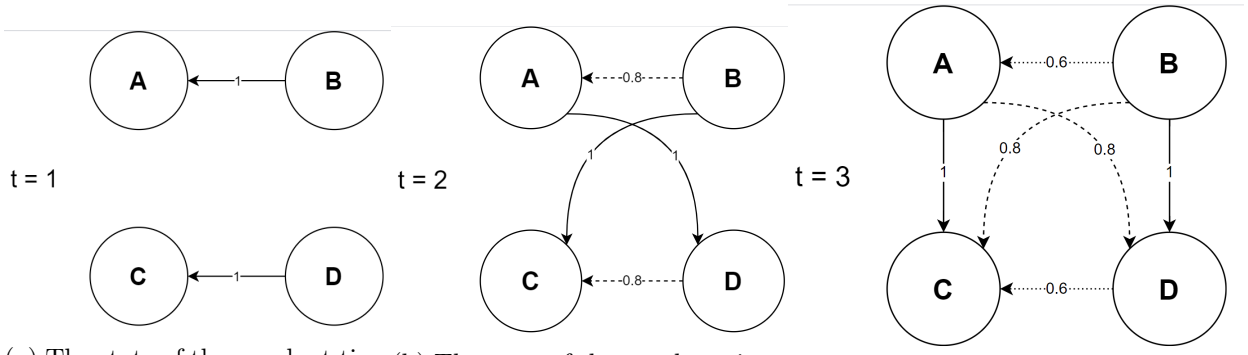
$$w^t_{e_{ij}} = e^{-(t - t_prev)}. \tag{4.2}$$

In practice we used the computation by a linear function, because values of the exponential function were too "negative" – the values for even not so distant matches went rapidly to zero and had practically no effect on the representation.

In order to simulate the ranking system between the teams, the propagation of information should only go one way. In the implementation of this work, the propagation happens from a losing team to the winning team. By this strategy, the propagated data becomes the rival's strength, and it is added to the winner's strength. This approach's downside is that the mean of the "ranks" is not constant but gradually increasing. It could have been solved by propagating the values from a winner to the loser with a negative sign.

Running Example

A running example of how the simplified version of the GNN model should work is shown below. It shows a model without embedding, the features of the teams are saved in the nodes rather than being trained as parameters in the Embedding layer. This example is very similar to how the PageRank model works that was discussed in Section 3.1.4, in fact the GNN model can be understood as a generalization of the PageRank model. Suppose, there is a league of 4 teams that have to define their ratings through a GNN model. The change of the states of the graph is shown in Figure 4.4. At time t_1 two matches take place: in the first one, a team A wins over the team B and in the second one, a team C wins over the team D. This means that to the disconnected graph of 4 vertices (one for each team) two edges are added (one for each match). The edges are directed – they go from a losing team to the winning team. This state is shown in Figure 4.4a. In the previous time tick all teams had a starting score of 1. At the end of time 1 the score values of the teams B and D stay the same at this time (since they did not win



(a) The state of the graph at time $t = 1$.

(b) The state of the graph at time $t = 2$.

(c) The state of the graph at time $t = 3$.

Figure 4.4: The example of how a simple GNN works.

	t_0	t_1	t_2	t_3
A	1	2	2.8	3.4
B	1	1	1	1
C	1	2	3.8	9.2
D	1	1	3	6.24

Table 4.3: The progress of the scores of the individual teams in time for the running example.

over anyone yet) while the score values for the teams A and C are calculated according to the following formula:

$$s_i^t = s_i^{t-1} + \sum_{j|e_{ij} \in E(G)} w_{e_{ij}}^t \cdot s_j^{t-1}, \tag{4.3}$$

where s_i^t is a score for a team i at time t, $w_{e_{ij}}^t$ represents a directed edge from the team j (the loser of the match) to the team i (the winner of the match). This way the score of the team i is the weighed sum of the scores of all the teams i won over. The weight of the edges in this particular example set in the following manner: 1 for the match that ended at time t, 0.8 for the match that ended at time $t-1$ and 0.6 for the match that ended at time $t-2$. In the following time ticks there are matches between all the teams from this league. In the end the team C won in all of the conducted matches, A and D won in some of them and B won in none of them. The scores for the teams till the time t_3 are shown in Table 4.3.

The resulting scores reflect the number of games for the teams C and B and are expected. The scores for the teams A and D are different even though they had the same number of victories. The difference in the scores explains the quality of the victories (if the losing opponent was good) and the time of the victories.

Model Components

A diagram of the GNN model can be viewed in Figure 4.3b. The label-encoded names of all teams in the data set across all leagues and seasons are embedded into vectors of higher dimensions. Those vectors then serve as node features for the first ConvGNN Layer block. A ConvGNN

Layer block represents a combination of a ConvGNN Layer with an activation function and a possible dropout layer. There can be multiple ConvGNN Layer blocks stacked on top of each other. After the information between the teams is propagated, the last ConvGNN Layer block produces the learned representation for all the teams. However, to make a prediction for a match, the model only needs to know the representations of the Home team and Away team. After they are filtered out and concatenated the model is ready to predict the result of the game using stacked Linear Layer blocks. The output of the last Linear Layer block is then run through the softmax layer that scales it and generates the predictions for three of the game result classes: Home team victory, Draw, Away team victory.

4.3 Evaluation frameworks

Usually, when working with data in terms of statistical procedures, one has to assume that the data are independent and identically distributed, meaning that the samples are drawn from the population randomly and sequentially. The data is often shuffled to ensure that these groups are independent when the samples are randomly divided into groups. For example, if someone made a random poll of a Yes / No question across the country, then shuffled the answers and divided them into two groups, the groups will have roughly the same ratio of "yes" to "no" votes. However, when using time series data or data with a time component, one cannot assume that the samples are independent, but rather exactly the opposite – the assumption should be that the earlier samples affect the later samples. That leads to the assumption about the data set being only that the data should be independently distributed.

There are two frameworks that can be distinguished to train and evaluate a predicting model:

1. Fixed evaluation. The data is subsequently divided into three categories: a training data set, validation data set, testing data set. Training data is used for a model to learn the parameters, the validation data set is used to fine-tune the model's hyperparameters, the testing data set is employed to evaluate the actual performance of the model. The drawback of this approach is that the model "loses" the validation data, leading to the model's inferior performance when dealing with time series.

2. Continuous evaluation. This approach rigorously divides the data set only into two categories: the training data and the testing data. The model is training gradually from a portion of the training data set captured from a window sliding with time and can be interpreted as a mini training data set. The model then runs through the mini training data set in mini-batches for multiple epochs. The validation accuracy and loss are computed from a portion of the data following the mini training data set. After the training and validation are performed, the window controlling the data moves, and the process starts again. An example of this process can be found in Figure 4.5.
   The testing can be done in two ways. One way is two fix the testing set and evaluate the

model on it at once. Another way (which will be called "sliding testing set" for future referencing) is evaluating the model on a small portion of the testing data, retraining the model using this small portion, and then working this process until the end of the training set. This way of testing is preferred since this approach simulates the real-life conditions – the prediction for the upcoming event is generated, after the match, the model is updated, the model generates the prediction for the next match.

Later in this work, the continuous evaluation will be used since it can utilize the latest data into the prediction.

Finally, the configuration of the Linear layer was taken from the ANN with an embedding model for training a GNN model.

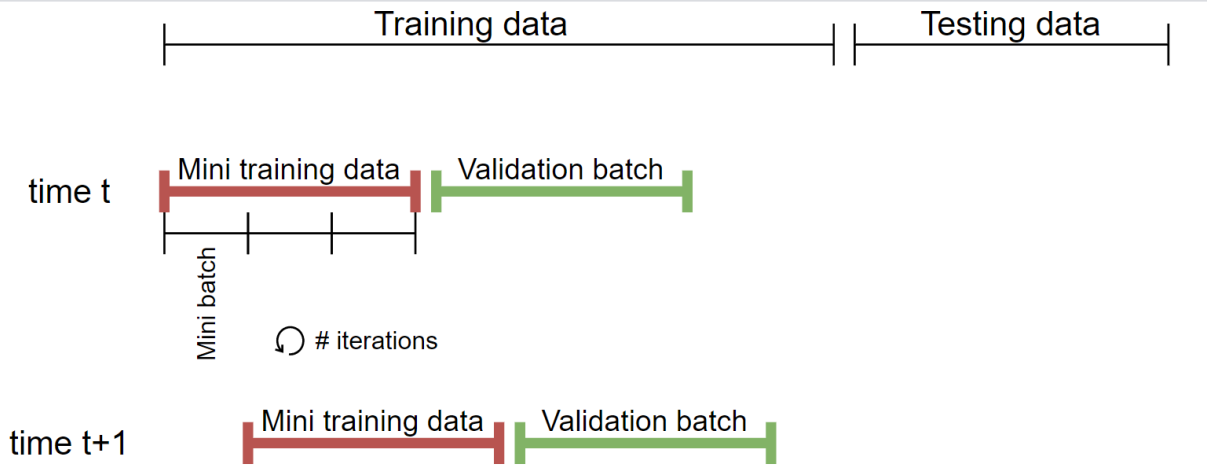


Figure 4.5: Example of the continuous evaluation. The red section represents the sliding window for capturing the training data.

4.4 Evaluation

4.4.1 Common setup

To be able to fairly compare the models, some common parameters had to be fixed. The parameters were settled after some tests and empirical evaluation of the results.

- Mini batch size: 9. For most leagues, this is the number of matches in the season played between a team's two successive games (there are 16-20 teams in a league, so if they start to play with each other, there will be 8-10 played games before a team plays the second time).
- Optimizer is fixed to be Adam [5].
- Epochs: 30. This value was set to be the initial number of epochs.

- Window size for capturing the training data: 40 mini-batches.
- Window size for capturing the validation data: 10% of the whole data set. Such a number was chosen to estimate the validation accuracy and loss robustly.
- Window size for capturing the testing data: 1 mini-batch.
- Dropout rate was set to 0.1.
- The learning rate mode refers to one of two approaches of setting the learning:
  1. The learning rate is fixed throughout the training phase.
  2. The learning rate decreases with time and is computed at time t for each slide of the sliding window as follows:

$$\lambda^t = \lambda_{base}(1-\gamma)^{k/bs/ws}, \tag{4.4}$$

     where λ_{base} is the initial setting of learning rate discussed above, γ represents the discount factor of the learning rate (it was set to 0.2), k is the ordinal number of the first match in the sliding window, bs stands for the batch size, ws represents the number of batches after which the learning rate should be decreased. $\lambda_{base}, \gamma, ws$ can be considered hyperparameters.

  The second approach is often used when training Graph Neural Networks [49]. However, in this case, the adaptable learning rate did not improve the model's performance. Relying on this conclusion, the learning rate remained constant over time for all models.
- A subtraction layer instead of a Concatenation layer was tested in both models. However, with both models, the performance was much poorer than when using a Concatenation layer.

4.4.2 Learned Hyperparameters

Learned hyperparameters include:

- The number of epochs for fitting data captured from a later sliding window. One approach is leaving the number of epochs fixed in time. However, to emphasize the importance of the later data on the predictions, the model can be exposed to the later data a higher number of iterations. To achieve that, the number of epochs was computed for every slide of the sliding window at time t according to the following formula:

$$n_{epochs}^{t+1} = n_{epochs}^{t} + log_{lb}k, \tag{4.5}$$

  where lb stands for logarithm base and was set to 1.5, k is the ordinal number of the first match in the sliding window. The logarithm base can be considered another hyperparameter for tuning.

- The possible values of the learning rate were limited to 0.01, 0.001, 0.0001, 0.00001.
- The maximal possible number of Linear layers was restricted to 5, minimal possible number was set to 2.
- The space of possible settings for the Linear Layers consisted of the following configurations:
  $(4,4,4,4,4), (8,8,8,8,8), (16,16,16,16,16), (32,32,32,32,32), (64,64,64,64,64),$
  $(4,8,16,32,64), (4,8,4,16,4), (32,64,32,128,4), (16,32,4,16,8), (16,4,16,4,16),$
  $(64,128,64,128,8), (128,128,128,128,128)$, where the number represents the number of neurons in a layer.
  During the validation process, it became evident that the models with a constant number of neurons in all layers have a better performance than the once in which the number of neurons changes from layer to layer.
- The maximal number of ConvGNN layer was set to 3, the minimal – to 1.
- Possible configurations of the ConvGNN layer included:
  $(1,1,1), (4,4,4), (8,8,8), (16,16,16), (64,64,64), (128,128,128), (128,64,32), (32,32,32),$
  $(32,16,8), (16,8,4), (128,32,8), (64,32,4), (4,64,4), (8,128,8),$
  $(8,128,64), (64,32,64)$.
  As well as in the Linear layer blocks, the constant number of neurons showed much better performance than with the changing one.
- The choice of activation functions was restricted to
  1. ReLu:
  $$ReLu(x) = max(0, x) \tag{4.6}$$
  2. hyperbolic tangent function: $tanh(x)$
  3. Leaky ReLu:
  $$LeakyRelu = max(\alpha x, x), \tag{4.7}$$
  where α was set to 0.1.

4.5 Implementation

The models were implemented in Python using the libraries PyTorch [50], and PyTorch Geometric [51]. The data was imported and examined using the Pandas library data frames [52]. The resulting models are saved into the binary files using a *pickle* library in Python [53].

The codebase for this work can be found in the attachment and in the public repository [2]. The code is divided into several files:

[2] `https://github.com/perevale/matches_prediction.git`

- DataTransformer: uploads the input data set, label encodes the teams, divides the data sets into training, validation and testing data sets;
- Dataset: creates the Data instances for all the leagues in the data set;
- Trainer: trains, validates, and tests the models directly;
- GNNModel, FlatModel: create the GNN and Simple ANN models;
- visualization: helps to visualize the training and embedding;
- utils: provides the utility functions for training.

The fine-tuning of the models' hyperparameters was executed on the remote server using the grid search. The scripts for this are located in the folder *scripts*.

4.5.1 Implementation procedure

The implementation was not linear, and there were multiple dead ends and complications. The most comprehensible procedure could be approximated as follows:

- The Simple ANN model is created and trained in a fixed manner. An earlier version was built in *Keras*, but later transferred to *PyTorch* because of *PyTorch's* higher flexibility and transparency.
- A simple GNN model is constructed, the model does not have any Linear layers, the predictions are produced directly by subtracting the two teams' representations and taking the sign of the output. Although the model was able to achieve over 50% of the training accuracy, it did not generalize well, and testing accuracy was close to random. In this model, there was only one trainable parameter in the ConvGNN Layer; instead of the edge weighing, there was a manual weighing of the nodes. This model was a starting point for the resulting GNN model. The implemented version is the same as reviewed in the running example in Section 4.2.2.
- The GNN model is meant to be a combination of the Simple ANN model and the Simple GNN model. It was first trained in a fixed way, then the retraining on the validation set was also applied. The features in the nodes were then changed to the embedding representation, and the edge weights were applied.

The models can be viewed in the attachment.

Chapter 5

Results

The following chapter is dedicated to reviewing the obtained results. The first section is going to examine the best achieved results. After that, there are going to be a review of some alternations to the best models and their effect on the models' performance.

5.1 Best models

To correctly evaluate the models, several variations of the data sets had to be taken into consideration. There are two data sets discussed in Section 4.1.2. However, both of them were rather large, and training one model on the soccer data set took up to three days. To fine-tune most of the hyperparameters, only one randomly chosen league from the soccer data set was chosen – GER1. It is represented by the name "1 league soccer".
The Berrar, Lopes, Davis, *et al.* [18] also chose only 26 "bigger" leagues out of 52 for the test set of the 2017 Soccer Prediction Challenge. They are represented by the name "chosen leagues soccer".
The hockey data set is represented by the name "NHL".
The comparison of the best-achieved models performance for the two approaches on the four data sets can be viewed in Table 5.1.

Table 5.1 shows that the model was able to fit the smallest of the four data sets far better than the larger ones. It makes sense intuitively: since the model only predicted the results for 36 teams from one league and in the next case of all leagues of soccer, the model had to be far more general to be able to fit the data of 1951 teams and finding the best possible param-

	1 league soccer	all leagues soccer	chosen leagues soccer	NHL
ANN with embedding	0.4962	0.4553	0.4894	0.4222
GNN	0.5231	0.4640	0.4926	0.4493

Table 5.1: Best achieved models' accuracy.

eters is more complicated if not impossible – different parameters work best for different leagues.

The table also proves the superiority of adding a graph structure over the linear layer for predicting the outcome of sports matches and displays that such an approach indeed helped to learn a better representation of teams in all four cases.

5.1.1 1 league soccer data set

The best configuration of a simple ANN model with an embedding layer is the following:

- 3 linear layers.
- Each layer consists of 4 neurons.
- The dimension of input embedding is 3.
- The learning rate is set to 0.001.
- The Dropout layers were placed after each Linear layer with the dropout rate of 0.1.

A model that best fit the data of the GER1 soccer league consisted of

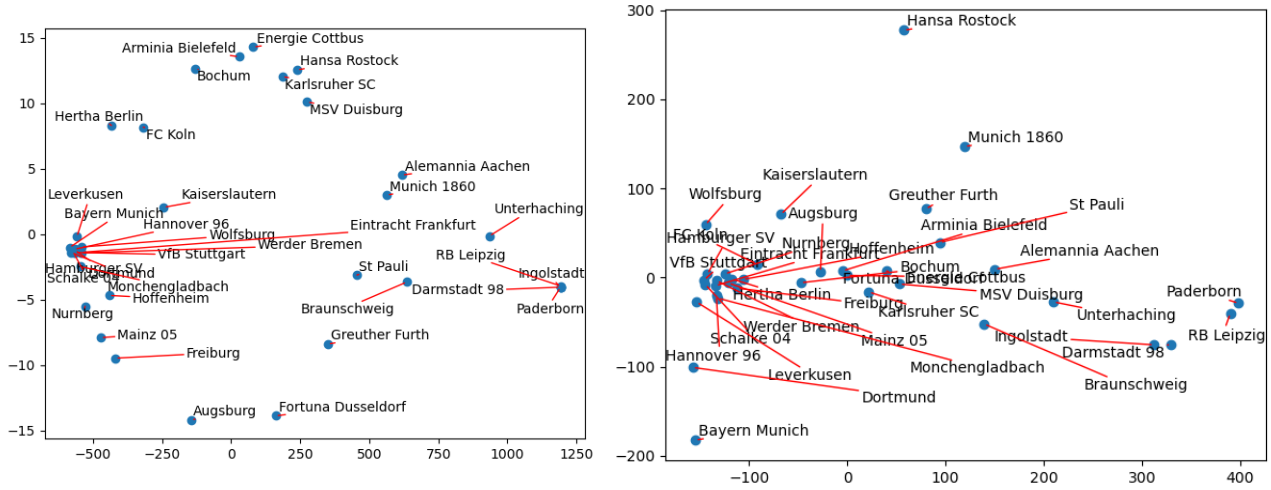
- 2 graph convolutional layers of dimension 4.
- Convolutional layers: GraphConv.
- The dimension of input embedding is 3.
- The learning rate is set to 0.001.
- The Dropout layer was placed after the first ConvGNN layer with the dropout rate 0.1.

It can be noticed that a relatively small number of neurons was needed for both of those models.

The resulting embedding can be viewed in Figure 5.1. It was obtained using PCA projection with two components. One can notice that the representation of the teams before the training is chaotic. The teams do not seem to be ordered. However, the embedding after the training shows us that they are represented in some arranged manner. Most teams are lined up in a noticeable order, which means that the horizontal axis might represent the ranking of the teams. The *Dortmund* and *Bayern Munich* are obviously the leaders since they are the left-most teams. After comparing some simpler rankings with the one in question in the table 5.2, one can assume that the vertical axis shows how sure the victories were for the teams throughout the timeline (the lower, the more sure is the victory) – *Bayern Munich* got the highest victories to losses ratio, and it is located in the bottom meaning that the victory is sure for the team. The PCA projection of the embedding with 1 component and its comparison to the simpler, more explicit ranking models – the difference between the number of won and the number of lost matches and the Elo rating 3.1.2 – is shown in the table 5.2.

Embedding ranking		Win-Loss Difference Ranking		Elo ranking	
Dortmund	-157.43285	Bayern Munich	303	Bayern Munich	154735.3
Bayern Munich	-154.72981	Dortmund	154	Leverkusen	71330.37
Leverkusen	-153.28636	Schalke 04	107	Schalke 04	68776.49
VfB Stuttgart	-145.66075	Leverkusen	106	Dortmund	58952.95
Schalke 04	-144.4327	Werder Bremen	55	Werder Bremen	43354.39
Wolfsburg	-144.27525	VfB Stuttgart	29	Wolfsburg	30104.49
Hamburger SV	-142.38426	RB Leipzig	13	Hertha Berlin	28068.66
Eintracht Frankfurt	-134.10876	Wolfsburg	8	RB Leipzig	3805.212
Hannover 96	-133.24687	Hamburger SV	-4	Fortuna Dusseldorf	-334.153
Werder Bremen	-133.03867	Hertha Berlin	-5	Hoffenheim	-465.699
Monchengladbach	-131.78793	Unterhaching	-7	Alemannia Aachen	-611.405
Nurnberg	-124.64906	Alemannia Aachen	-9	Braunschweig	-963.602
Mainz 05	-121.12248	Hoffenheim	-10	Ingolstadt	-1886.1
Hertha Berlin	-117.85539	Paderborn	-10	Greuther Furth	-2311.5
Freiburg	-112.02846	Fortuna Dusseldorf	-11	Hamburger SV	-2482.05
Hoffenheim	-106.32832	Munich 1860	-12	Unterhaching	-3267.8
FC Koln	-91.78417	Ingolstadt	-14	Paderborn	-3313.84
Kaiserslautern	-67.60042	Karlsruher SC	-15	MSV Duisburg	-3689.42
Bochum	-47.461567	Braunschweig	-15	Darmstadt 98	-4890.84
Augsburg	-27.295141	Greuther Furth	-17	Karlsruher SC	-5344.51
Arminia Bielefeld	-4.547561	Augsburg	-20	St Pauli	-5715.27
Energie Cottbus	0.06291556	Mainz 05	-20	Augsburg	-9752.47
Karlsruher SC	21.897686	Darmstadt 98	-21	Arminia Bielefeld	-11341.6
Fortuna Dusseldorf	40.037266	MSV Duisburg	-25	Kaiserslautern	-11988.6
MSV Duisburg	53.142437	Monchengladbach	-27	Mainz 05	-12581.5
Hansa Rostock	57.575123	St Pauli	-29	Munich 1860	-15113.2
Greuther Furth	79.802826	Kaiserslautern	-33	VfB Stuttgart	-16361.5
St Pauli	93.84798	Arminia Bielefeld	-39	Freiburg	-19590.7
Munich 1860	119.45913	Hansa Rostock	-41	Hansa Rostock	-24236.3
Braunschweig	139.13191	Bochum	-42	FC Koln	-29456.9
Alemannia Aachen	150.2522	Hannover 96	-47	Bochum	-30939.8
Unterhaching	209.98987	Energie Cottbus	-49	Energie Cottbus	-32774.2
Ingolstadt	312.1592	Freiburg	-56	Hannover 96	-41534
Darmstadt 98	329.73306	Eintracht Frankfurt	-64	Nurnberg	-42820
RB Leipzig	389.7234	Nurnberg	-66	Eintracht Frankfurt	-43121.1
Paderborn	398.24176	FC Koln	-67	Monchengladbach	-46239.8

Table 5.2: Comparison of the three approaches to encode the ranking of the German League. The main trends, are the same for all the three rankings, e.g. *Bayern Munich*, *Dortmund*, *Leverkussen* are at the top in all the rankings. The difference resides in the mapping of the teams with poorer performance: for example, *FC Koln* occupies the positions in the middle of the rating and in the other rankings it is found in the bottom. This can be influenced by the fact that the embedding better captures the time trends in the data.



(a) The embedding representation of the GER1 league before training the GNN model.

(b) The embedding representation of the GER1 league after training the GNN model.

Figure 5.1: Embedding of the GER1 league.

5.1.2 All leagues soccer data set

The best simple ANN model for the soccer data set was the following:

- 5 linear layers.
- Each layer consists of 16 neurons.
- The dimension of input embedding is 3.
- The learning rate is set to 0.0001.
- The Dropout layers were placed after each Linear layer with the dropout rate 0.1.

The best simple GNN model for the soccer data set was the following:

- 3 graph convolutional layers of dimension 8.
- Convolutional layers: GraphConv.
- The dimension of input embedding is 3.
- The learning rate is set to 0.0001.
- The Dropout layer was placed after the first ConvGNN layer with the dropout rate 0.1.

The number of neurons in these models is slightly larger than for the models trained for one league.

5.1.3 Hockey data set

The best simple ANN model for the hockey data set was the following:

- 5 linear layers.

- Each layer consists of 4 neurons.
- The dimension of input embedding is 10.
- The learning rate is set to 0.00001.
- The Dropout layers were placed after each Linear layer with the dropout rate of 0.1.

The best simple GNN model for the soccer data set was the following:

- 3 graph convolutional layers of dimension 1.
- Convolutional layers: GraphConv.
- The dimension of input embedding is 10.
- The learning rate is set to 0.0001.
- The Dropout layer was placed after the first ConvGNN layer with the dropout rate 0.1.

The number of neurons is smaller than in the case of the soccer data set model, however, the number of layers is maximal possible – 5 Linear layers and 3 graph convolutional layers.

5.2 Weighed models

As can be seen from Section 4.1.2, both data sets have more matches won by home teams. That is a known phenomenon called home advantage. The circumstances that may affect the sway in the home team's favor can be psychological (such as local fans' support, known grounds, subconscious referee's bias) and physiological (visiting team's fatigue from traveling, different climate or time zones). Team advantage is even measurable: Marek and Vávra [54] propose a comparison method for home advantage based on Jeffrey Divergence, which is a symmetric adaptation of the Kullback-Leibler divergence. There is an even smaller percentage of the matches ending with the draw. That can be explained by the fact that if the teams feel that their game end with a draw, they tend to play riskier. Another reason for this anomaly is that penalty shootouts can be appointed during competitions as specified by the rules or directed by a referee. The shootouts continue until one team scores, and another one does not. This measure is often used as a tiebreaker at the end of the seasons or competitions.

As a result, some models give preference to the home team in their predictions as well, and infrequently predicts the draws. To overcome this, the optimizer can punish incorrect predictions unequally for different classes. Mistakes of the less populated class are penalized proportionately to the ratio of the number of samples in the class with the highest population N_hpc over the same number in the class in question N_a:

$$w_a = \frac{N_{hpc}}{N_a}. \tag{5.1}$$

This approach is better in this case than undersampling highly populated classes, since the latter technique might deprive the model of some insights from the time component of the data.

	Draw prediction	Away win prediction	Home win prediction
True draw	6	39	**80**
True away win	2	67	**85**
True home win	1	36	**205**

(a) The confusion matrix before the weighing.

	Draw prediction	Away win prediction	Home win prediction
True draw	29	**52**	44
True away win	30	**68**	56
True home win	50	61	**130**

(b) The confusion matrix after the weighing.

Table 5.3: The confusion matrices before and after weighing the the optimizer to penalize the incorrect prediction proportionately to the classes' population.

The confusion matrices before and after the discussed weighing can be seen in Table 5.3. It can be noticed that after the weighing, the model stopped giving clear preference to the home team.

The results for all data sets are shown in Table 5.4. If compared to Table 5.1, it can be noticed that the models' performance after the weighing is somewhat lower than the best achieved so far. That was the reason why the models with reported performance were penalized for incorrect predictions equally for all classes. If the strength between two teams is very similar, the outcome of the match becomes almost random since the influence of other factors unmentioned in the data set enhances, and they become game-changing. It will be further discussed in Section 6.

5.3 Choice of ConvGNN Layers

There are many Graph Convolutional layers available. Three of them were selected and discussed in Section 2.3: ChebConv, GCNConv, GraphConv. These three types of ConvGNN Layers are the most popular and were used in the multiple scenarios with GNN [11], [13]. The effect of different Graph Convolutional layers on the models' performance is shown in Table 5.5. Table 5.5 shows that spatial ConvGNN Layers are more suitable for this task than the spectral ones.

	1 league soccer	all leagues soccer	NHL
weighed ANN	0.4483	0.4259	0.4082
weighed GNN	0.4968	0.4561	0.4183

Table 5.4: The models' accuracy after the weighing.

	1 league soccer	all leagues soccer	NHL
GraphConv	**0.523076923**	**0.46401423**	**0.449328**
GCNConv	0.51962	0.46004	0.4396
ChebConv	0.52018	0.46193	0.44532

Table 5.5: Models' performance with different graph convolutional layers.

5.4 Choice of a testing set

The effect of different choices of testing sets on the models' resulting accuracy can be viewed in Table 5.6. Table 5.6 shows that the model indeed utilizes the time component for the predictions

	1 league soccer
Sliding testing set	0.523076923
Fixed testing set 10%	0.519230769
Fixed testing set 15%	0.461533434
Fixed testing set 20%	0.456653149

Table 5.6: Experiments with different testing sets.

– the further the match is from the last update of the model, the worse is the quality of the prediction. The continuous evaluation framework proved to be the best fit for this kind of task.

5.5 Comparison with the bookmaker

5.5.1 Bookmaker's predictions

Decimal bookmaker's odds represent the amount of money won for every unit wagered and can be interpreted as the inverted probability estimation for each of the results of the game. To deal with the fact that the bookmaker adjusts the odds for personal gain and as a result, the inverted probabilities of the three outcomes sum up to more than one, the actual predicted probability of victory of the home team H can be obtained after normalizing the inverted bookmaker's odds as in the formula:

$$P(H) = 1 - \frac{BO_H}{BO_H + BO_A + BO_D}, \tag{5.2}$$

where the BO stands for betting odds.

The data for bookmaker's predictions were taken from a website [45].

5.5.2 Comparison of the models

Figure 5.2 shows the inferior performance of the acquired model compared to the state-of-the-art model Xgboost [38] and the market odds on the soccer data set. The poorer performance compared to the bookmakers' predictions is expected since the bookmaker operates with a much more substantial set of relevant features than just the history of the games' outcomes. The lower accuracy, when compared to the Xgboost model, might be induced by the absence

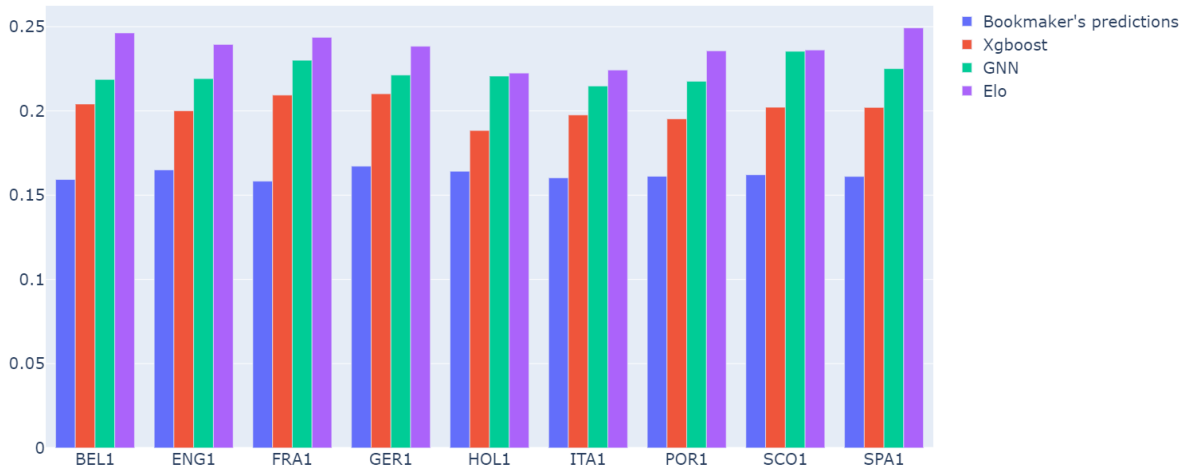


Figure 5.2: The comparison of performance between the bookmaker's predictions, the Hubáček, Šourek, and Železný [38] Xgboost model and a GNN model trained within this thesis work. The vertical axis shows the RPS and should be read "the lower RPS, the higher the accuracy".

of the explicit rating scores, such as pi-ratings. It can also be seen that when compared to the trained Elo rating model, the GNN model has somewhat elevated performance for most leagues. The metaparameters for the Elo model were chosen the following:

- $\gamma = 1.44$;
- $k = 2.9$;
- $c = e$;
- $d = 400$.

The meaning of the metaparameters was discussed in Section 3.1.2.

5.6 Discussion of the results

During the experiments, the proposed models showed satisfactory results, however, there were several observations that going to be discussed below. The advantages and disadvantages of the proposed models are going to be reviewed in the end of this section.

5.6.1 Viability of the proposed models

Working with the soccer data set was problematic due to the high number of unconnected leagues: a model's performance was drastically changing throughout the leagues. Using such a high number of disjointed leagues for one model seems unnecessary and even objectionable in real life since it is preferable to find a model that represents each league in the best possible way rather than a model that works only satisfactorily for all the leagues.

Since the different parameters work best for different leagues, the more general models showed inferior performance, when compared to the models that were trained only on one league in soccer. This even raises the question about the viability of such a general model – it might be useful for giving an approximate estimation of the result, however, models fit on more specific data give much better predictions of a match.

5.6.2 Contributions of the proposed models

The advantage of the suggested models is the fact that they are domain-independent and is a more general approach than the ones discussed in Section 3.
Another benefit of the discussed approach to sports prediction is that the trained models can be used as a rating model as well. The representations of the teams are easily extractable, and the respective ranking can be obtained by using dimensionality reduction algorithms, such as PCA.

The main disadvantage is the fact that the model has multiple hyperparameters that can influence the performance of the model, and although their behavior was tested on multiple sports, generally, their optimal value has to be found experimentally.
Another possible shortcoming of the approaches discussed in work is computational and time demands during the training phase of the algorithm that are much higher when compared to the ones of the various rating systems. Rating systems usually traverse through the data only once, while the neural methods discussed in this work used the sliding window for data capturing and displayed the data to the model for multiple iterations. However, this overhead is typical for all advanced models discussed in Section 3.2.
One more downside of the created models emerges when using them as rating models. The justification for the representations of the individual teams is not as transparent as in the case of the rating models, where the rules that give the score to the teams are created manually. In this case, the score of the team may be affected by its own playing behavior as well as the behavior of the teams that are considered neighbors in the spatial graph structure that covers all the teams. Nonetheless, the discussed approach is not entirely opaque because the general rule bases on the history of the past games' results only and can be easily formulated as "the more a team wins over stronger opponents, the higher is its score". This rule is common for all the models discussed in Section 3.

Chapter 6

Conclusion

Predicting the results of sports matches is a challenging task: Berrar, Lopes, and Dubitzky [55] warn that gaining more complex data does not necessarily lead to better predictions of sport results because the narrow margin of victory is inherent in most professional matches in all sports. Additionally, some details that have a high impact on the game's outcome, such as penalties, red cards, are not induced by the teams' strength, but rather by the quality of refereeing, weather, and athletic field conditions, intrateam communication, human factors. Due to those aspects having short-term influence and being somewhat random, it can be challenging to capture and keep track of them. However, their effect can be game-changing .

6.1 Goal Fulfilment

In this work, I have collected a significant amount of the data necessary to create a predictive model. The data consisted of two data sets containing the history of the results for the sports matches for soccer and ice hockey. Then I have discussed the related works that include the rating systems, statistical models, and several more advanced models that used machine techniques. I have proposed two solutions that might help to predict the results of the matches. The idea behind it was to learn the representations of the individual teams and then use that representation as input features to a prediction model. In the first solution, the teams' representations were learned using an embedding layer. In the second solution, the embedding solution was complemented by a spatial graph structure where the vertices were the individual teams, and the edges represented the past matches. The edges were weighted differently to reflect the fact that the more recent matches should have more effect on the team representation. The best prediction came from a GNN model trained on one league of the soccer data set with the accuracy of 52.31%. The model trained on a chosen number of leagues also showed a good accuracy of 49.62%, which is a result comparable with the state-of-the-art model [38]. The models showed a better performance when compared to the results obtained from the Elo ratings (RPS of 0.2176 vs. 0.2362). That showed that a more complex representation structure indeed helped

to achieve better predictions of the outcome of future matches. The experiments with the testing set proved that the so-called sliding testing set is the most suitable type of framework for testing the model. Besides, this type of framework is a close approximation of how the model might be used in real life. After that, multiple choices of the most popular ConvGNN layers were explored and tested. The experiments showed that the GraphConv layer brought the best results for all three experiments. The reason for this might be the fact that the spatial ConvGNN is a more suitable layer than the spectral ones. An attempt to balance the class representation in the data sets did not produce any positive results for any of the data sets. Although in the current state, the models are not ready to compete with the market odds or the state-of-the-art models, the conducted experiments demonstrated the viability of the representation learning, and the insights from this work can later be used in further researches.
That can lead to a conclusion that all the requirements from the thesis assignment are satisfied.

Appendix A

Attachment structure

The attachment contains the following:

- README.md: has instructions for running the code;
- *scr* folder with the source code;
- *scripts* folder with the scripts for model training;
- *data* folder with the input data;
- *models* folder contains the models trained during the work.

Bibliography

[1] R. Lanciani, "Gambling and cheating in ancient rome", *The North American Review*, vol. 155, no. 428, pp. 97–105, 1892, ISSN: 00292397. [Online]. Available: http://www.jstor.org/stable/25102412.

[2] J. Drchal, *Statistical machine learning (be4m33ssu) lecture 5: Artificial neural networks*, 2017. [Online]. Available: https://cw.fel.cvut.cz/old/_media/courses/be4m33ssu/anns_ws2017.pdf.

[3] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic differentiation in machine learning: A survey", *CoRR*, vol. abs/1502.05767, 2015. arXiv: 1502.05767. [Online]. Available: http://arxiv.org/abs/1502.05767.

[4] *Intuitive understanding of backpropagation*. [Online]. Available: https://cs231n.github.io/optimization-2/.

[5] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[7] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012, ISBN: 0262018020.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013. arXiv: 1301.3781 [cs.CL].

[9] V. Raunak, V. Gupta, and F. Metze, "Effective dimensionality reduction for word embeddings", in *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 235–243. DOI: 10.18653/v1/W19-4328. [Online]. Available: https://www.aclweb.org/anthology/W19-4328.

[10] (). Word embeddings, [Online]. Available: https://www.tensorflow.org/tutorials/text/word_embeddings.

[11] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks", *IEEE Transactions on Neural Networks and Learning Systems*, 1–21, 2020, ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2978386. [Online]. Available: http://dx.doi.org/10.1109/TNNLS.2020.2978386.

[12] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, *Weisfeiler and leman go neural: Higher-order graph neural networks*, 2020. arXiv: 1810.02244 [cs.LG].

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, 2017. arXiv: 1606.09375 [cs.LG].

[14] G. Arfken, *Mathematical Methods for Physicists*, Third. San Diego: Academic Press, Inc., 1985, pp. 731–748.

[15] *Source code for torch$_g$eometric.nn.conv.cheb$_c$onv*. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/cheb_conv.html.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", *CoRR*, vol. abs/1609.02907, 2016. arXiv: 1609.02907. [Online]. Available: http://arxiv.org/abs/1609.02907.

[17] *Source code for torch$_g$eometric.nn.conv.gcn$_c$onv*. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/gcn_conv.html.

[18] D. Berrar, P. Lopes, J. Davis, and W. Dubitzky, *The 2017 soccer prediction challenge*, 2017. [Online]. Available: https://osf.io/ftuva/.

[19] E. S. Epstein, "A scoring system for probability forecasts of ranked categories", *Journal of Applied Meteorology and Climatology*, vol. 8, no. 6, pp. 985 –987, 1Dec. 1969. DOI: 10.1175/1520-0450(1969)008<0985:ASSFPF>2.0.CO;2. [Online]. Available: https://journals.ametsoc.org/view/journals/apme/8/6/1520-0450_1969_008_0985_assfpf_2_0_co_2.xml.

[20] A. C. Constantinou and N. E. Fenton, "Solving the problem of inadequate scoring rules for assessing probabilistic football forecast models", *Journal of Quantitative Analysis in Sports*, vol. 8, no. 1, 12 Mar. 2012. DOI: https://doi.org/10.1515/1559-0410.1418. [Online]. Available: https://www.degruyter.com/view/journals/jqas/8/1/article-1559-0410.1418.xml.xml.

[21] C. Leung and K. Joseph, "Sports data mining: Predicting results for the college football games", *Procedia Computer Science*, vol. 35, 2014. DOI: 10.1016/j.procs.2014.08.153.

[22] M. J. Maher, "Modelling association football scores", *Statistica Neerlandica*, vol. 36, no. 3, pp. 109–118, 1982. DOI: 10.1111/j.1467-9574.1982.. [Online]. Available: https://ideas.repec.org/a/bla/stanee/v36y1982i3p109-118.html.

[23] C. Ley, T. V. de Wiele, and H. V. Eetvelde, "Ranking soccer teams on the basis of their current strength: A comparison of maximum likelihood approaches", *Statistical Modelling*, vol. 19, no. 1, pp. 55–73, 2019. DOI: 10.1177/1471082X18817650. eprint: https://doi.org/10.1177/1471082X18817650. [Online]. Available: https://doi.org/10.1177/1471082X18817650.

[24] O. Hubacek, G. Sourek, and F. Zelezny, "Score-based soccer match outcome modeling–an experimental review", *MathSport International*, 2019.

[25] (). United states chess federation, [Online]. Available: http://www.uschess.org/content/blogsection/12/35/ (visited on 02/02/2020).

[26] A. E. Elo, *The Rating of Chessplayers, Past and Present*. New York: Arco Pub., 1978, ISBN: 0668047216 9780668047210. [Online]. Available: http://www.amazon.com/Rating-Chess-Players-Past-Present/dp/0668047216.

[27] L. M. Hvattum and H. Arntzen, "Using ELO ratings for match result prediction in association football", *International Journal of Forecasting*, vol. 26, no. 3, pp. 460 –470, 2010, Sports Forecasting, ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2009.10.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169207009001708.

[28] (). Aoe3 simple elo ladders, [Online]. Available: http://aoe3.jpcommunity.com/rating2/ (visited on 02/04/2020).

[29] A. C. Constantinou and N. E. Fenton, "Determining the level of ability of football teams by dynamic ratings based on the relative discrepancies in scores between adversaries", *Journal of Quantitative Analysis in Sports*, vol. 9, no. 1, pp. 37 –50, 30 Mar. 2013. DOI: https://doi.org/10.1515/jqas-2012-0036. [Online]. Available: https://www.degruyter.com/view/journals/jqas/9/1/article-p37.xml.

[30] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.", Stanford InfoLab, Technical Report 1999-66, 1999, Previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/.

[31] A. Y. Govan, C. D. Meyer, and R. Albright, "Generalizing google's pagerank to rank national football league teams", 2008.

[32] J. Shi and X.-Y. Tian, "Learning to rank sports teams on a graph", *Applied Sciences*, vol. 10, no. 17, p. 5833, 2020.

[33] D. Berrar, P. Lopes, and W. Dubitzky, "Incorporating domain knowledge in machine learning for soccer outcome prediction", *Machine Learning*, Aug. 2018. DOI: 10.1007/s10994-018-5747-8.

[34] A. Tsokos, S. Narayanan, I. Kosmidis, G. Baio, M. Cucuringu, G. Whitaker, and F. Király, "Modeling outcomes of soccer matches", *Machine Learning*, vol. 108, no. 1, pp. 77–95, 2019, ISSN: 1573-0565. DOI: 10.1007/s10994-018-5741-1. [Online]. Available: https://doi.org/10.1007/s10994-018-5741-1.

[35] S. M. Arabzad, M. Tayebi Araghi, S. Sadi-Nezhad, and N. Ghofrani, "Football match results prediction using artificial neural networks; the case of iran pro league", *Journal of Applied Research on Industrial Engineering*, vol. 1, no. 3, pp. 159–179, 2014, ISSN: 2538-5100. eprint: http://www.journal-aprie.com/article_43050_3a948ff1c96f37f39f16ec113decc317.pdf. [Online]. Available: http://www.journal-aprie.com/article_43050.html.

[36] R. Nyquist and D. Pettersson, "Football match prediction using deep learning", 2017.

[37] P. Pugsee and P. Pattawong, "Football match result prediction using the random forest classifier", in *Proceedings of the 2nd International Conference on Big Data Technologies*, ser. ICBDT2019, Jinan, China: Association for Computing Machinery, 2019, 154–158, ISBN: 9781450371926. DOI: 10.1145/3358528.3358593. [Online]. Available: https://doi.org/10.1145/3358528.3358593.

[38] O. Hubáček, G. Šourek, and F. Železný, "Learning to predict soccer results from relational data with gradient boosted trees", *Machine Learning*, vol. 108, no. 1, pp. 29–47, 2019. DOI: 10.1007/s10994-018-5704-6.

[39] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", *CoRR*, vol. abs/1603.02754, 2016. arXiv: 1603.02754. [Online]. Available: http://arxiv.org/abs/1603.02754.

[40] S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik, "Gradient-based boosting for statistical relational learning: The relational dependency network case", *Machine Learning*, vol. 86, no. 1, pp. 25–56, 2012, ISSN: 1573-0565. DOI: 10.1007/s10994-011-5244-9. [Online]. Available: https://doi.org/10.1007/s10994-011-5244-9.

[41] A. C. Constantinou, "Dolores: A model that predicts football match outcomes from all over the world", *Machine Learning*, vol. 108, no. 1, pp. 49–75, 2019, ISSN: 1573-0565. DOI: 10.1007/s10994-018-5703-7. [Online]. Available: https://doi.org/10.1007/s10994-018-5703-7.

[42] Sourav, *Top 10 most popular sports in the world*, 2020. [Online]. Available: https://sportsshow.net/top-10-most-popular-sports-in-the-world/.

[43] M. Coolman, *Sports and identity: Case study czech republic and ice hockey*, 2010.

[44] W. Dubitzky, P. Lopes, J. Davis, and D. Berrar, "The open international soccer database for machine learning", *Machine Learning*, vol. 108, pp. 9–28, 2018.

[45] *Football results, statistics amp; soccer betting odds data.* [Online]. Available: `https://www.football-data.co.uk/data.php`.

[46] *Hockey/usa: Nhl 1998/1999.* [Online]. Available: `https://www.flashscore.com/nhl-1998-1999/results/`.

[47] M. R. Izadi, Y. Fang, R. Stevenson, and L. Lin, *Optimization of graph neural networks with natural gradient descent*, 2020. arXiv: `2008.09624 [cs.LG]`.

[48] S. Dabhi and M. Parmar, *Nodenet: A graph regularised neural network for node classification*, 2020. arXiv: `2006.09022 [cs.SI]`.

[49] J. Park, D. Yi, and S. Ji, "A novel learning rate schedule in optimization for neural networks and it's convergence", *Symmetry*, vol. 12, no. 4, p. 660, 2020.

[50] [Online]. Available: `https://pytorch.org/`.

[51] *Pytorch geometric documentation¶*. [Online]. Available: `https://pytorch-geometric.readthedocs.io/en/latest/index.html`.

[52] *Pandas.dataframe¶*. [Online]. Available: `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html`.

[53] *Pickle - python object serialization¶*. [Online]. Available: `https://docs.python.org/3/library/pickle.html`.

[54] P. Marek and F. Vávra, "Comparison of home advantage in european football leagues", *Risks*, vol. 8, no. 3, p. 87, 2020.

[55] D. Berrar, P. Lopes, and W. Dubitzky, "Incorporating domain knowledge in machine learning for soccer outcome prediction", *Machine Learning*, vol. 108, no. 1, pp. 97–126, 2019. DOI: `10.1007/s10994-018-5747-8`. [Online]. Available: `https://app.dimensions.ai/details/publication/pub.1106040005andhttps://link.springer.com/content/pdf/10.1007/s10994-018-5747-8.pdf`.