Deep Learning-Based Price Prediction for Online Game Trade Markets

eingereichte
Bachelorarbeit
von

Mario Semmler
Matrikel-Nr.: 1322915

geb. am 20.08.1994

Universität Augsburg
Lehrstuhl für
Embedded Intelligence for Health Care and Wellbeing

Shahin Amiriparian

Betreuer: Manuel Milling

Abstract

Contents

Chapter 1

Motivation

In the course of digitization the computer games industry has gained more and more recognition over the last years. The number of games, that are played in competitions, awarded with price money, is constantly rising.
One of those games is FIFA
FIFAs competitive online mode Ultimate Team made up 28% of the companies total net revenue of the year 2019. [1] Annually it brings in 800 million dollar. [2]
This chapter describes the motivation of the underlying topic.
- nice to know how much a new player is worth. - is there a rational connection between the attributes and the price?
The worth of a new player on the transfer market is on the one hand side an interesting information for the gamers, as they can decide more effective if they should buy a player already or wait until his price has leveled. On the other hand side it is very useful for the company itself to see what effect the insertion of a new specific player could have.

Chapter 2

Introduction to Fifa Ultimate Team

This chapter provides an introduction to the most played online mode of EA Sports yearly released computer game FIFA: FIFA Ultimate Team (commonly called FUT). We will have a closer look at the current game FIFA20 in the following. The underlying game concept remains throughout the last years released games and will be explained in this chapter to provide a basic understanding of the game. This years FIFA20 is played by more than 10 million people worldwide, which shows the relevance and impact of the game and makes it even more interesting to research towards different fields. [3]
The following section does not claim to be comprehensive in terms of covering all the information about FIFA Ultimate Team. We will only observe and explain facts about the game mode that are relevant for the striven price prediction.

2.1 Structure of the game mode

FIFA is a football simulation video game, that basically offers to play with a team consisting of real football players against other gamers. We will from now on call the football players in the game *players* and other persons you can play against *gamers* to avoid misunderstandings.
2009 the FIFA series received the online mode FIFA Ultimate Team. The main idea behind this game mode is to build up your own dream team with the players you would like to play and compete with other gamers teams. The players are represented by cards that you receive as depicted in Figure 2.1.
There are different ways to obtain those cards. The two main possibilities are either receiving them from packs, which include a random selection of those cards, which can be bought with real money or buying them on the virtual transfer market of the game with *coins*, which is the name of the in game currency of FUT. Coins can be gained in different ways. You get coins for every match you play, as rewards for participating in different competitions and selling cards on the transfer market.

Figure 2.1: This figure shows some cards that represent the players in FUT

Besides the players cards there are also other types of cards as for example consumables, that can be applied to the players, like position changes. [4]

2.1.1 Chemistry

When it comes to building your squad there are a few things that have to be taken into account. One of them is the *chemistry*.
The main idea behind this catchword is that you need to build your squad in a way where the players in some kind are related to each other. Being related means in this context, that two players are from the same nation, are playing in the same league or even play at the same club.
If we have a look at figure 2.2 one can see that a team is set up in a certain formation, which you can choose freely, and that there are connections between the players. Every player has a maximum chemistry value of 10 and a minimum value of 2. The higher the chemistry, the better is the player when you play him in the game.
Therefor we can say, it should be aimed to build a team where every player has a chemistry value of 10. To obtain a chemistry value of 10, a player needs to have the same position written on its card as the position you put him in the certain formation. This means for example, that on should not put a goalkeeper card into a striker position.
The second important point is about the other players that are next to/ connected to the player. The relation to another player is rated from 0 to 3 points. A player

needs as many points/links as connections to other players.
As an example we have a look at the player on the top left position (left ST (striker)) in Figure 2.2. This player needs 3 links in total due to his connection to the left CM (central midfielder), the CAM (central attacking midfielder) and the right ST. The player is obviously supplied with enough links, as he is already connected by a link with the value of 2 to both other players with the english nation from the same league (premier league).
[5]

Figure 2.2: This figure shows a FUT20 squad of a certain formation

2.1.2 Attributes

This section will cover all information about the players attributes like pace, shooting, passing and so forth. These attributes describe what makes a player unique. We distinguish between in game stats, that define the performance in the game and info attributes like nation, height, type of the card, etc.

In game stats:
The in game stats are all values between 0 and 99, the higher the better, and are clustered into 6 categories: pace, shooting, passing, dribbling, defending and physicality.
You can see the whole list of in game stats in figure **??**.
We considered the in game stats to have a huge impact on the price of a player

as they obviously correlate directly with the performance of the player in a match, what is one of the biggest points about the game, hold a strong performing team to beat other gamers.
An interesting question about those attributes is, if the importance and relevance is as homogeneous as their value distribution or if there are more important and less important attributes among them. We will come back to this later at the point were we are analysing the weighting of our trained model.

Info attributes:
Those attributes contain as already mentioned different types of information about a player with a wide range of values. From plain text over a selection about leagues, nations and clubs to 1 to 5 rated attributes we find a lot of different values. (Maybe add information or insert list of all attribute infos) It is possible that some of them are not even affecting the price of player, because nobody takes the information into account and others might have a bigger impact than expected. We will also have a look on this in section 5.4.

Those two attribute groups contain all the data that we use to define our features to train and develope our models.

2.2 Different Card Types

As already mentioned FUT provides a digital representation of the players, that are playing in the football leagues in the "real world", through obtainable cards. There are different card types.
At the start of the game in September 2019 there were only three base card types: bronze, silver, gold. Every of those three card types holds rare and common cards, where the common are more likely to be packed from the card packs one can buy or receive through mentioned game modes. The fourth kind of cards in the beginning of FIFA20 are icon cards, that represent former football stars like Pele, Ronaldinho and Michael Ballack.
The special property of those icon cards is, that they provide a chemistry link to every card, no matter which league or nation. If another card additionally matches the nation of the icon card, it obtains two links from the icon card. This means, an icon is not dependent on the other cards regarding its chemistry value and furthermore is helpful to link other cards in the team. Icon cards have therefor a very helpful additional use next to their in game performance value, which increases their demand.
Throughout the year, EA provides several new card types, which contain new versions of already existing players that are higher rated and better. They are released during themed events, that take place for one to two weeks most at the times. One of the most expected events is for example the "Team of the year" (TOTY), where

the best eleven players of the last season are getting a major boosted version of themselves in the game.
Cards that are released during a event are only available to be packed during the time of the event, whereas the "normal" four card types are brought into the game during the whole season. Those new cards have a big impact on the market because they are requested very highly and are only available for a short time. This often leads gamers to sell their older version of a card or other cards of their club to be capable of buying the new cards, which impacts the market heavily.

2.3 The transfer market

The transfer market is the place in FUT where all the players can be sold and bought. The price of the players results of supply and demand. A very important fact in this suspect is that every player has a price range, determined by the publisher EA Sports, with a minimum and a maximum value. This price range is not always consistent and is sometimes adjusted by EA Sports which makes the the minimum and maximum value not a reliable and constant attribute for our prediction. [6]
It should be mentioned, that a player can be bought and sold unlimited times, but for every transaction of an player, EA keeps 5% of the in game currency as taxes. This means that if one gamer sells a player successfully for 100.000 coins to another one, he receives only 95.000 coins. This concept mainly reduces inflation but also makes it more different for people who try to trade and draw profit by selling players at a time for cheap price and sell them on another for more coins. [7]

2.4 Summary

FIFA Ultimate Team (FUT) is the most played game mode in the yearly released game FIFA by EA Sports. The main point is about competing with other gamers by playing virtual football with a self constructed team of recreated real football players in different competitions of the game.
The players a gamer owns are represented by cards, that are stored in his virtual club. There are multiple different types of cards and also different versions of a single player, represented by different cards.
There are some restrictions for the teambuilding, to make it more challenging and demand more creativity when it comes to develope a team.
Players can be traded between the gamers on the games internal transfermarket by buying them with the in game currency called coins.
After this short introduction to the world of FUT we proceed with our theoretical basement of our prediction approach, that we later apply to the case of FUT.

Chapter 3

Problem formulation

We chose machine learning, in particular the subfield deep learning, as our main approach to solve the problem of time-independent price prediction in terms of the game FIFA 20 and it's game mode FIFA Ultimate Team. Today machine learning is a widely researched scientific field, that constantly gains in relevance and interest. This leads to the fact, that it is split into many different research areas with varying approaches and targets. We will use a fully connected deep neural network to solve the task of price prediction of football players in EA Sports game FIFA Ultimate Team.
To get feedback about the quality of our obtained results of the deep learning approach we also applied other models to the same problem. In the subsequent sections we will discover the theoretical base of those different approaches, where we focus the most on the last section about our main concept, the deep learning model.

The subsequent assumptions hold for all of the whole chapter:

- x is a vector of n input feature values of one data sample like in our specific case the attributes of one football player.

- X denotes the set of m training sample features that are used to train our models. This means $X = \begin{bmatrix} x_1 & ... & x_m \end{bmatrix}$, where m is the number of training data samples.

- y describes the target value of one sample, in our instance equal to the price of a specific football player.

- Y designates the set of m training sample target values, that are used to train our models, this means $Y = \begin{bmatrix} y_1 & ... & y_m \end{bmatrix}$

3.1 Problem description

We first want to define and understand what task we are facing in our case of the price prediction of FIFA Ultimate Team football players. Being more precise we want to make the best possible time-independent prediction of the price of a random football player from the online trade market in the game FIFA Ultimate Team by considering various information about this specific player, the so called features. This description of our task matches very well with the definition of a regression problem.
"[...] using data to predict, as closely as possible, the correct real-valued labels of the points or items considered." [8]
The task of regression can be described by the target of finding a function $r : x \to y$ where X and y are the at the beginning defined notations.
As a difference to the very popular classification task we are not aiming to predict the correct label of the sample but try to minimize the resulting error between prediction and label. This error is called loss and is denoted by $L : y \times y' \to \mathbb{R}_+$
One of the most popular loss functions is the squared loss function, defined by: $L_2(y, y') = |y - y'|^2$.
As L only returns the error for one specific prediction, we want to transfer our obtained results and merge it into a more general formula with more significance, taking into account all training samples.
Therefore we use the following equation, called the mean squared error:

$$R(r) = \frac{1}{m} \sum_{i=1}^{m} L(r(x_i), y_i) \tag{3.1}$$

m denotes the number of single training samples and r and L are the former defined functions.
We have now a regression function r that returns a label value y for an arbitrary input-vector X, a loss function L that measures the magnitude of error between the predicted label and the actual label value and the function $R(r)$ that hands back the mean squared error of our regression function and the training data. [8, p. 237-238]

3.2 Linear Regression

At first we have a look at a well known basic approach when it comes to regression tasks, the linear regression.
The main idea is to find a function $f(x)$ that approximates the related y value to a certain x vector as close as possible for all $x \in X$. We define this function by $f(x) = w \cdot \Omega(x) + b$, where w is meant to be an adjustable parameter to make the model fit the label values y the best and $\Omega : X \to \mathbb{R}^m$, which transforms the input feature matrix into a m-dimensional vector to be combinable with linear shape of

the model.
$f(x)$ returns y' as a prediction value for the related y value of x. As discussed in the former section we can use L_2, the squared loss, to evaluate the error of the prediction of $f(x)$. The idea is to minimize the squared error for all $x \in X$, taking into account all training samples simultaneously, like we did in the former section where we defined the mean squared loss. The squared loss can be applied to our optimization by transforming it into the following formula:

$$\min_{\mathrm{W}} F(W) = \frac{1}{m}\|X^T W - Y\|^2 \tag{3.2}$$

where X are our features denoted as $\begin{bmatrix} \theta(x_1) & \dots & \theta(x_m) \\ 1 & \dots & 1 \end{bmatrix}$ with an additional 1 because of dimensional reasons when multiplying it with the weights vector. W are the weights written as vector in the following form $\begin{bmatrix} w_1 \\ \dots \\ w_N \\ b \end{bmatrix}$. Because F is a convex, differentiable function it has a global minimum if $\nabla F(W) = 0$. Thus we can reformulate equation 3.2 as following:

$$\frac{2}{m}X(X^T W - Y) = 0 \Leftrightarrow XX^T W = XY \tag{3.3}$$

The obvious unique solution of the right side of equation 3.3 if XX^T is invertible is: $(XX^T)^{-1}XY$. Otherwise there is a family of solutions, defined by the pseudo-inverse of matrix XX^T.
Linear regression can be easily visualized for the simple case of n=1 which corresponds to the case of one input feature for each x.
In figure 3.1 we can see this visualization, where the blue points our data points (x and y) and the red line is the linear approximation that minimizes the mean squared error.

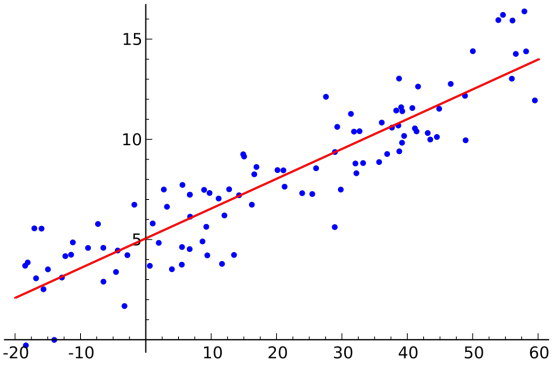

Figure 3.1: linear regression problem []

To close this section about linear regression we wanna have a short look at the expense of this approach.
As explained it takes a computation of the inverse of a matrix which is a computation task of a complexity of approximately $\Omega(n^{2.4})$ to $\Omega(n^3)$. This is equivalent to a time extension of the computation by a factor of 5.3 to 8. []

3.3 Support Vector Machine

"[...] one of the most theoretically well motivated and practically most effective classification algorithms in modern machine leraning: Support Vector Machines (SVMs)." [9, p.63]
Support vector machines are often utilized to solve classification problems where a data set of training samples with labels is used to determine a hyperplane for the given classification problem like in figure 3.2. The gained hyperplanes are then applied to the test samples to classify them.

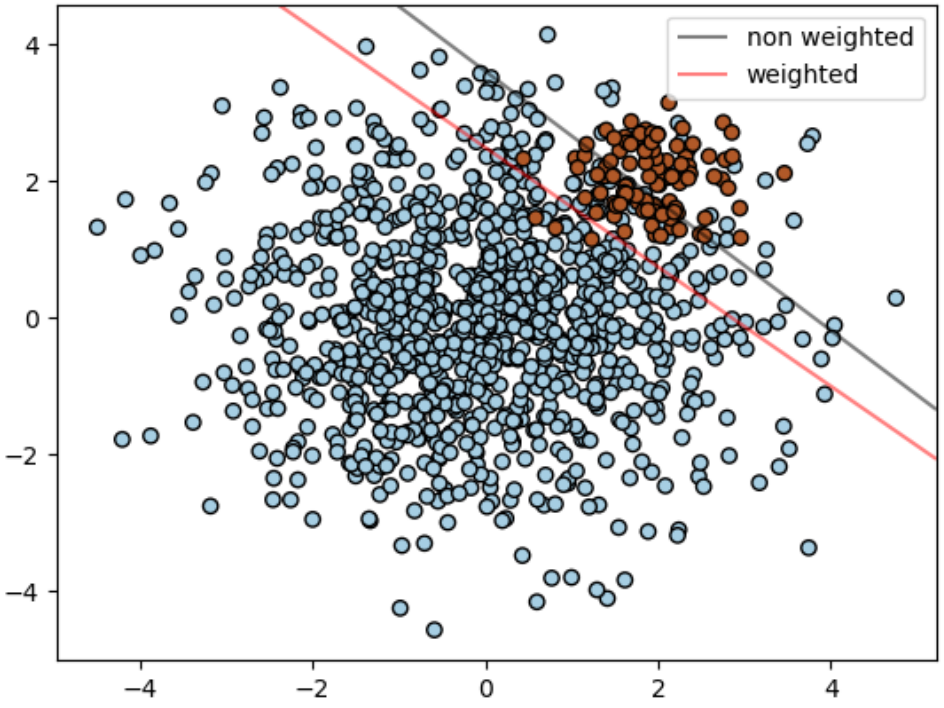

Figure 3.2: SVM hyperplane determination [?]

To implement the SVM on our data we used a python library called scikit-learn [10]. The library is open source, free to use and offers a lot of possibilities from classifi-

cation over regression up to other useful scientific tools.

3.3.1 Support Vector Regression

Even though the SVM is more popular for its classification capacity, it is also capable of solving regression problems. The regression problem can actually be defined as a generalization of the classification problem.
First we want to have a closer look at the classification task to understand how the transition to the regression works.
The most simple classification task is the binary classification task. This problem can be categorized as a convex optimization problem, where it's the task to find the maximum margin separating hyperplane, that classifies as many training points correctly as possible. This optimal hyperplane is represented by support vectors.

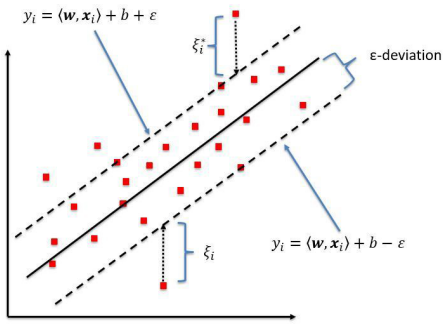

Figure 3.3: Example of the ϵ-tube around f(x)

Now the generalization to a regression problem is obtained by an ϵ-insensitive region around the pure classification function, called ϵ-tube. [11]
This means we want to find a function that does not deviate more than ϵ for every target value of the training data and simultaneously is as flat as possible. The former mentioned flatness of the curve can be assessed by minimizing the norm: $||w||^2$
This leads us to the following formulated optimization task:

$$\text{minimize } \frac{1}{2}\|w\|^2 \tag{3.4}$$

$$\text{subject to } \begin{cases} y_i - (w \cdot x_i + b) \leq \epsilon \\ (w \cdot x_i + b) - y_i \leq \epsilon \end{cases} \tag{3.5}$$

Minimizing w and at the same time ensuring that the error is smaller than ϵ, leads in some cases to very few solutions in other cases to no solution at all. To cope with this challenge and also allow some errors in given cases, we introduce another

concept, called *slack variables* which allows us to reformulate 3.4 to the subsequent, feasible optimization problem:

$$\text{minimize } \frac{1}{2}\|w\|^2 \tag{3.6}$$

$$\text{subject to } \begin{cases} y_i - (w \cdot x_i + b) \leq \epsilon \\ (w \cdot x_i + b) - y_i \leq \epsilon \end{cases} \tag{3.7}$$

[12]

3.4 Random Forest

The *random forest* algorithm is a another regression method, based on the foundation of *decision trees*.
A decision tree is an ordered tree with a root r. For the edges $(v, x_1), ..., (v, x_n)$ from a nonleaf vertex v, we call $x_1, ..., x_n$ the *decisions* at v. An arbitrary x_i is denoted as the ith *choice* at v and x_{i+1} is called the *next choice* after x_i. [13, p. 39]
This means every vertex of a decision tree represents r an issue and every edge from this vertex is a possibly outcome to the issue. The next vertex is another issue and so on. At the bottom of the tree every leaf represents an output value for the superordinated issue the decision tree is meant to decide.
A decision tree can be created from a data set with multiple feature values and one target value. It is built up from the root to the leafs. This means one starts with the determination of which feature should be the first vertex. This question can be answered by checking how good every single feature performs on independently predicting the right value for the regression problem for all samples of the data set. This idea is continued for all the following stages/vertexes until all features have been used.
The transition from one decision tree to random forest happens by applying at first the concept of ensemble learning. This means creating multiple decision trees and aggregating their results. The aggregating for regression is implemented by taking the average of all the predicted value. Secondly it uses bagging (bootstrap aggregation), where only a random subset of all samples of the data set are used to construct every decision tree. The concept of bagging targets to improce stability and accuracy of the algorithm. This method so far is already a possible approach of solving a regression task on its own.
Random forest is modifying this concept by making the construction of the trees even more random. This is obtained by taking only a subset of the features available for determining a new vertex of a decision tree at every stage. This additional specification makes the algorithm very robust against overfitting. [14]
- only 2 parameters and also very insensitive to the parameters → user friendly

3.5 Deep Learning

As already mentioned in the beginning of the section machine learning in general is a strongly rising researched field of science with many different subfields adapted to different research questions.
The title of this section and the whole thesis tells that we are concentrating on the section called Deep Learning.
The question "What is Deep Learning?" brings us to the recognition, that there are many different answers to that question. [15]
What the most of definitions have together is, that deep learning is about learning relations between multiple features/variables by using various layers to extract and transform those features.
The adjective *various* indicates the intention of keeping the boundaries fluid to ensure not being narrowed by restrictions while researching. It is often agreed to talk about a deep neural network if the network consists of three layers or more - one input and output layer and any number greater than zero of hidden layers in between. [16, p. 263]
The further-reaching question about deep neural networks we want to cope with is about what makes them unique and more advantageous than other machine learning approaches.
We have already seen, that deep learning is about the depth of the neural network that is used.

3.5.1 General architecture

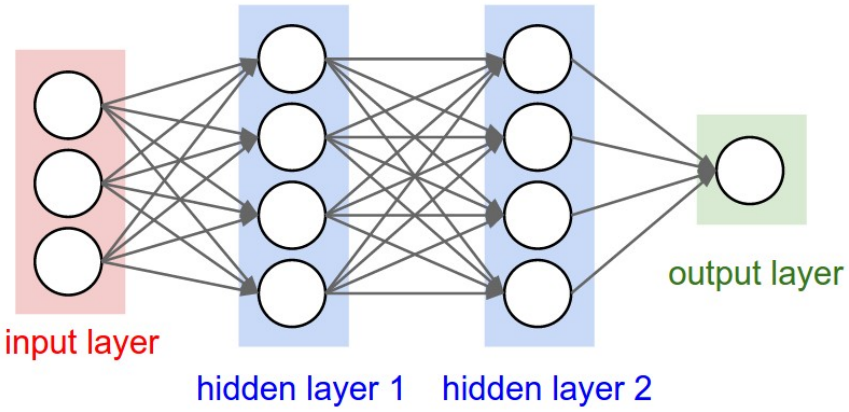

Figure 3.4: Fully connected neural network with 2 hidden layers

The idea of artificial neural networks (ANN) comes from biological models like the human neural system. Already the human cortex is structured in multiple layers of biological neurons.
Taking the nature as an example, further research in the field of machine learning

resulted in building deep neural networks (DNN) that can cope with very complex related data of enormous size. [16, p. 257]
In general a deep neural network consists of an input layer that takes in all the feature values of the data and has usually as many neurons as the number of features. The input layer is succeeded by an arbitrary amount of hidden layers, where the amount of neurons can't be determined by certain rules, but is part of the research and development process of any individual problem. In the end we apply an output layer, that has exactly one neuron for regression problems.
One example for a specific architecture of a neural network is the fully connected neural network (FCNN), which is defined by the structure of a connection from every neuron to all neurons of the next layer as we can see in figure 3.4.

3.5.2 Gradient Descent

"Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems." [16, p. 113]
The Gradient Descent algorithm is an often used method for optimizing the weights in neural networks. The basic idea is to measure the gradient of the error function, to find out which "direction" minimizes the error the most. The next move is to make a small step, defined by the learning rate, into this descending direction. If the gradient turns out to be zero, the error has reached a local minimum of the error function.
The learning rate is a very important hyperparameter in machine learning and can be to high and also to low. In case of a learning rate that is to high it is possible to miss out minima because the gradient has changed within distance of one step. Otherwise a to small learning rate can result in a very slow and inefficient training progress. (MAYBE INCLUDE FIGURE OT BOTH CASES?)

3.5.3 Backpropagation

Now we want to have a look at how the weights in a DNN are being optimized. Therefor we are having a closer look at the concept of *Backpropagation*.
The breakthrough in developing a method of properly training DNN or neural networks in general was achieved inter alia with an article published in 1986. The article with the title *Learning Internal Representations by Error Propagation"* [**?**] introduced the concept of backpropagation.
The main idea about it is that for every training instance that is fed into the neural network, the output of the last layer is compared to the known target value. It is traced back how much of the resulted error is caused by each neuron of the last hidden layer.
This procedure is continued by exploring the source of the error of each neuron in the last hidden layer from the preceded layer and so forth until we end up at the

input layer.
With this sequence we measure efficiently the error gradient throughout all the weights of the neural network. By now there hasn't been any progress or optimization for the neural network. To effectively learn from the measured errors the backpropagation algorithm does a Gradient Descent step using the obtained information about every single neuron and its weight.

3.5.4 Batch Normalization

To face a very popular problem when it comes to neural networks and especially training them, the *vanishing gradients* problem, we introduce the concept of *Batch Normalization*.
The vanishing gradients problem can be roughly described by the effect, that gradients are often getting smaller when it comes to the lower layers. This leads to the incident of very low change to no change in the weights of the lower layers, which results in a lack of convergence of the training.
As brought up already, we have a solution to this problem.
Batch Normalization was proposed by Serfey Ioffe and Christian Szegedy in one of their papers in 2015 called "Batch Normalization: Accerlerating Deep Network Training by Reducing Internal Covariate Shift" [] and addresses amongst others the mentioned problem.
Roughly speaking Batch Normalization is an additional operation, that is added to a layer, which learns the layer the optimal scale and mean of its input.

Chapter 4

Deep Learning application

4.1 The data set

At first we have a look at the underlying data set. As mentioned before our source is the whole database of players that are featured in FIFA20 and additionally all the special card versions of the players at the timestamp of 22.04.2020.
We decided to choose only the gold cards, which are defined by having a rating of 75 and higher, due to the fact, that only those players are bought to be played. The silver and bronze players mostly serve the purpose of completeness. Most of them can be bought by the minimum price, therefor the only thing the neural net could learn or figure out would be that the price of the silver and bronze players are not influenced by their in game performance. But this is a obsolete information, that we already know.
The collected features can be separated into two groups. On the one hand side the attributes of the player that are rated from 0 to 99, as for example the low level attributes like *Acceleration*, *Dribbling* or *Jumping*, the superior summary one level up like *Pace*, *Shooting* etc. and the overall rating, that represents the quality of the player in a single top level rating value. We can see the list of all *attribute*-features in figure 4.1
The second group of features is the collection of all the remaining information stats that is heterogeneous in their value. For example the feature *nation* holds the information about the nationality of the player, whereas *weak foot* describes the level of ability of a player to shoot with is weaker foot by a value between one and five. The list of all features in this category can be seen in figure **??**

In the following some facts about the data set:
- In total 3882 players
- Snapshot of the players database on 22.04.2020
- Average label value / price of 119.833
- Smallest label value: 350
- Highest label value: 10.880.000

- Distribution of the label value / price
- Number of features: 53 (36 in game stats, 17 other heterogeneous attributes

4.1.1 Data-Scraping from Futbin

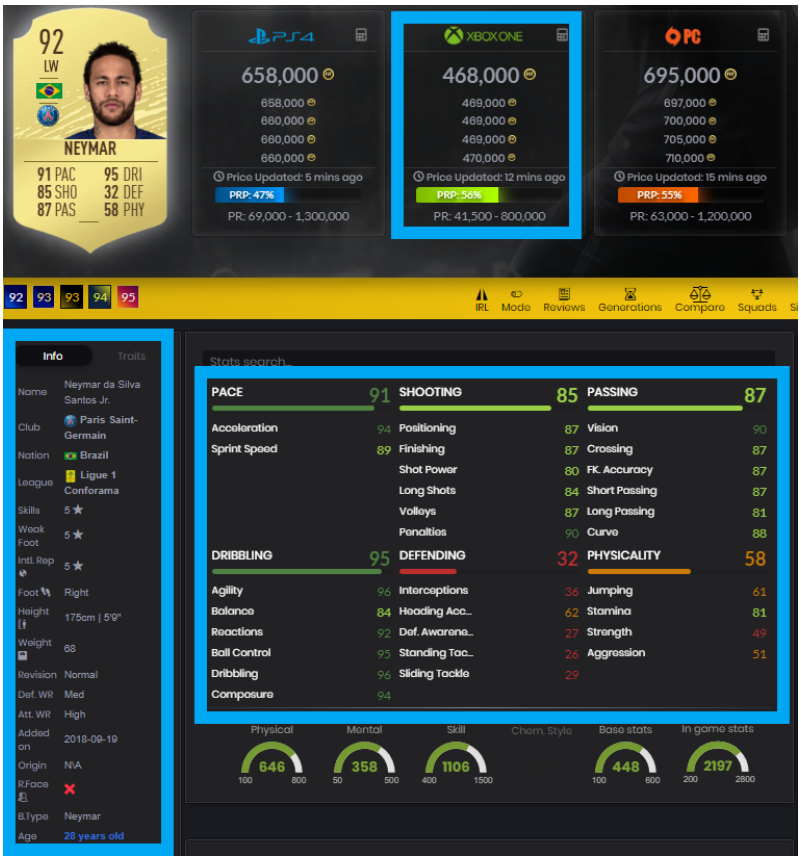

Figure 4.1: This figure shows the detailed player page on Futbin

Since the database of the players is not provided by EA Sports directly we had to built a workaround and scrape the data from Futbin, a website that offers the latest players database. [17]
We used a python based script to open the mentioned website with the framework selenium to navigate through the players list, read the data of the players directly from the html source code and write them into a local .csv file.
In figure **??** we can see the detailed page of a single FIFA20 player. We scraped all the data that is highlighted with the blue frame. On top we can see the five cheapest offers for the player on the certain transfer market.(differentiation between xbox, ps4 and PC market). Subsequent we see also the information when the data was collected.
The other two blue boxes cover attributes and the info features we mentioned earlier. The termination of the script took around 16 hours due to the fact, that we had

to enter each players detail page to collect the data. The used data was gathered between 13:18 on 21.04.2020 and 05:31 on 22.04.2020.

4.1.2 Processing the data

After scraping the data from Futbin we had to process the data before we could train and optimize our machine with it. Some of the steps were implemented within the .csv, but most of the work happened during runtime after the data is imported into the python application.

4.1.2.1 Preprocessing the csv

As mentioned in the previous chapter we collected always the three lowest prices that were indicated by Futbin. We have taken the average of those three prices to create our future target value.
Furthermore we had to remove certain kind of players that didn't fit the criteria. We started with a number of 3894 players. At first we sorted all the players out that were listed with a price of zero. A price of zero can be traced back to different reasons. For example the player can be extinct, what means nobody is currently offering the player on the market. In any case a price of zero is not useful for our experiment so we decided to remove those kind of players. After this reduction we had 3638 players left.
The next set of players we had to remove have been the goalkeepers due to the fact, that there attribute-features are different, what means, that they didn't fit into the scheme. Subtracting them left us with a final set of 3304 players.
The last change that we applied to the .csv-file was to randomize the data set. Therefor we used the =Rand() function of Microsofts Excel which returns a random value between 0 and 1. We inserted a random value in an extra column in every line and sorted the rows afterwards by the column with the random number from high to low.
The now obtained .csv file represents the base of all following tests and optimization approaches. A randomized dataset without zero-priced players and goalkeepers.
The average price value over all players is 119.388, the highest value is 10.880.000, the smallest value is 350. The price range for players on the transfer market has a upper limit at 15.000.000 and a lower limit at 350. []

4.1.2.2 Processing after import

Handcrafted Features:
We implemented a option to handcraft some of the features by own knowledge about the game to check if it helps the model to be fed data that is a bit more predefined already. For example the feature 'league' has more than 30 different possible values that are represented by a numeric value after the label encoding. To make this

information more suitable for the model we substituted each value by a value from 1 - 5, where 5 should indicate a higher price and 1 a lower price. We applied the same concept to the three features: position, league and nation. Those features are obviously important ones regarding the explanations about the chemistry in subsection 2.1.1.
The idea behind the substitution of the position feature values is, that there are positions, that are categorically more expensive than others. As already pointed out by the order of the leagues in the FUT20 menu there are top leagues, that contain the best and therefor most expensive players. The same concept exists for the nations. For the leagues for example, the top five leagues are: Premier League, LaLiga Santander, Serie A, Bundesliga, Ligue 1.

Labelencoding:
As mentioned previously one of the two groups of features is heterogeneous in their values. This means in this case they had not only another scale but we found also non-numeric values, that can't be fed to our model. We used a labelencoder to substitute the non-numeric values by numerical values. In particular we used a built in function of the pandas dataframe.

Feature set selection:
We implemented the option to choose different sets of features for the model train. The model should actually find out which features are more important and which features can be taken less into account, but we wanted to check if we can take away work from the neural net and speed up learning by preselecting a smaller set of features.

Preprocessing of the label value:
Another approach of preprocessing the data was testing the impact of normalizing and taking the logarithm of the label values. We used the built in function of *tf.keras.utils.normalize()* that we already used to normalize the feature values. For the logarithm approach we used the common logarith (logarith to the base of 10).

Train, devel and test split:
We also implemented the possibility to modify a parameter that set the percentage of data that was used as train, devel and test set. At first we applied the split between train and test set by simple python methods. The distribution to train and devel was handled by the *validation_split* parameter in the *fit()* function of the model.

Exponentiation of the in game stats :
Furthermore we wanted to see if it matters if we change the size of the steps between the in game stat values, that were initially values between 1 and 99, by potentiating them and therefore obtaining a non-evenly distributed set of values.

4.2 Code setup

In this section :
- code execution step by step:
⇒ Read csv file → preprocessing as explained in 4.1.2.2 → obtained x_tr y_tr fed into model function
⇒ train_and_evaluate_models() → adding layers to model → model .compile .fit .predict → return prediction_list
⇒ write_price_differences → export 1 and 2

4.3 Export from the model

We implemented three different exports to our model after training and the prediction on the test set. These exports were meant to help determining the parameters of the model and optimizing its prediction capability.
On the one hand side we used a framework called *TensorBoard* to visualize the data that we obtained during the training and validation stage. On the other hand side we simply exported data into csv-files to track the predictions that the model made.

4.3.1 Tensorboard export

TensorBoard is a visualization toolkit that can be used with Tensorflow. We primarily used the trainings and validation loss recording and visualization feature for our purposes to gain a impression on the learning behaviour of our model for different parameter combinations. [18]
It helped for example to figure out when our model started to overfit, simply by checking the progression of the training and validation-loss curves. As we can see in figure 4.2 after approximately epoch 1500 - 1600 the training loss continues to drop but the validation loss starts to rise. This is an indicator of overfitting.

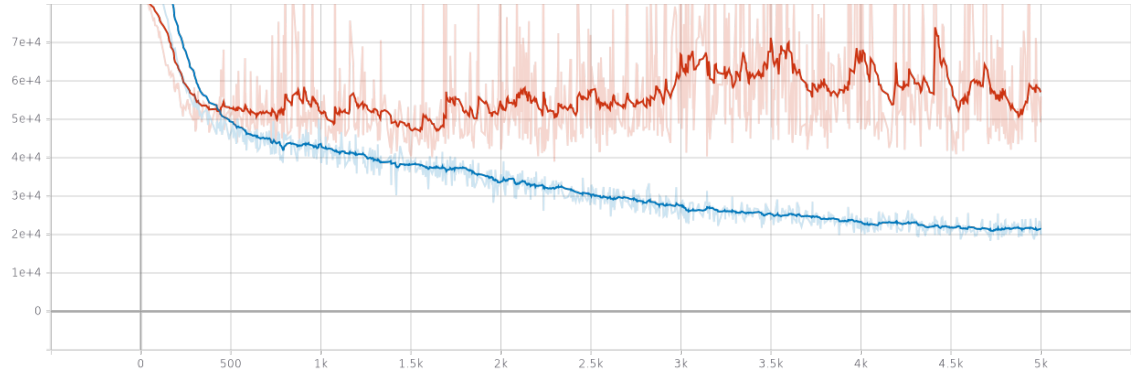

Figure 4.2: Example of a plot of training and validation loss by TensorBoard

Another feature you can see in figure 4.2 is the option of *Smoothing* the curve. In other words computing and presenting an averaged curve to get a better impression on how the curve looks like without the outliers.

4.3.2 RMSE and prediction export

The obtained testing data was written into two different kinds of csv-files. The first one stored for each training and fitting process the final mean absolute error (MAE) and the root mean squarred error (RMSE). It additionally provides the information about the parameter selection as we can see in figure 4.3.
We used mainly this export to make our parameter tests and develop the model with the best possible prediction results.

MAE	RMSE	name	feature_set	epochs	lr	batch_size	validation_split	optimizer	logarithm	normalize	batchnorm	exponent	train_split
70885	324580	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
70939	313358	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
74558	332975	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
69961	311347	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
78865	305935	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
73247	330326	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
78701	344740	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
71037	319984	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
73255	334890	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
72004	309060	standard	6	200	0.01	64	0.2	adam	FALSE	FALSE	FALSE	FALSE	0.7
91617	471186	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
86184	352927	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
88013	377944	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
80953	331522	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
76072	316770	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
105788	559322	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
125900	715003	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
168371	1099777	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
81531	323962	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
75614	309986	logarithmized	6	200	0.01	64	0.2	adam	TRUE	FALSE	FALSE	FALSE	0.7
66380	278678	batch-normalization	6	200	0.01	64	0.2	adam	FALSE	FALSE	TRUE	FALSE	0.7
67610	269860	batch-normalization	6	200	0.01	64	0.2	adam	FALSE	FALSE	TRUE	FALSE	0.7
64587	294670	batch-normalization	6	200	0.01	64	0.2	adam	FALSE	FALSE	TRUE	FALSE	0.7
65473	285863	batch-normalization	6	200	0.01	64	0.2	adam	FALSE	FALSE	TRUE	FALSE	0.7
61149	281727	batch-normalization	6	200	0.01	64	0.2	adam	FALSE	FALSE	TRUE	FALSE	0.7
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 4.3: Error export

The second export is more of a detailed prediction record and shows the predicted target value, the actual target value, the difference and all the features of the sample. This export served more as a second source to build a deeper understanding where the biggest prediction errors occur.
Of course this helped as well to determine and select some parameters more precisely but was taken into account only in second place after the MAE and RMSE export.

4.3.3 Hyperparameter determination

One of the main tasks in machine learning is the development of a well adjusted model to the given problem.
In figure 4.4 we can see the basic configuration that we took as the starting point for our tests. This base setup was developed by preceding sample tests to develop a sense for the given regression problem. We chose the MAE (Mean Absolute Error)

as our quality criterion and loss function, to determine which parameter works best. The training and test of the model for each parameter combination was executed ten times and was evaluated by the average of the ten resulting MAEs.
In addition we considered the training and validation loss curve for our decision, since under some circumstances it is important to watch the development of the curve and draw conclusions from it, i.e. discovering the incidence of overfitting.

Architecture	Activation-Function	Optimizer	Loss-Function	Used Feature Set
256-128-64-32-16	Relu	Adam	MAE	All features

Learning-Rate	train_split	Validation-Split	Batch-Size	Epochs
0.01	0.7	0.2	64	200

Batch Normalization	Dropout	Logarithm	Normalization	Exponent
FALSE	FALSE	FALSE	FALSE	FALSE

Figure 4.4: Parameter start allocation

Subsequent we are going through the different parameters we have tested and discussing the results of the test:

Target value preprocessing:
In section 4.1.2.2 we already dealt with the topic of preprocessing. The difference to the occurrence in this section is on the one hand side the data we are applying the preprocessing and on the other hand side the methods we use. After preprocessing the features in the mentioned former section we are now elaborating the effect of editing the **target values** before handing it to our neural net also called scaling.
For this purpose we compared three different approaches, the basic setting, normalizing the target values and taking the natural logarithm of the target values.
It turned out, that scaling by the logarithm obtained the best results as we can see in figure 4.5.

	Average MAE	Standard Deviation	Highest MAE	Lowest MAE
Base	79257	1408	81944	77225
Normalize	75191	3740	81863	70438
Logarithm	74222	6800	90332	68940

Figure 4.5: Test results - target value scaling

Figure 4.5 shows that in terms of the average MAE the logarithm is in the first place by an advance of 1000 coins, where it holds the lowest MAE value of a single training and test run by a gap of 2500 coins with an outcome of 68.940.
We noticed the higher standard deviation which was mainly caused by one big outlier at around 90.000. We also counted on stabilizing the model in later stages anyway.

Learning rate:
As our next parameter we wanted to elaborate the best learning rate for our model. Of course we have to consider that a smaller learning rate may imply a need of more epochs to reduce the loss to the same point as it can be reached by a higher learning rate.
But first we want to have a look at the obtained results of our first test. We used the values [1, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001] for our test and received the results displayed in figure 4.6.

Figure 4.6: Test results - learning rate

It turned out that XXX provided the best results as allocation for the learning rate parameter. By taking a look at the loss curve of this learning rate in comparison to the other learning rates we can also assume that even by training with more epochs XXX would reach the best results. To ensure that we made the test by running the training and testing again for a lot more epochs and as we can see in figure 4.7 the test confirmed the assumption.

Figure 4.7: Caption

Optimizer:
At first we want to talk about the difference between the three used optimizers. Gradient Descent is our first optimizer. It works straight as we have explored in section **??**. It is a very basic approach and does what it's supposed to do, but has its limitation in speed and efficiency.
As our second optimizer we chose the RMSProp. In the opposite to simpler optimizers RMSProp only accumulates the gradient of the most recent iteration in opposite to many others, that include all gradients from the start of the training. This leads to a almost always converging progress to the global minimum.
Our third optimizer is the Adam optimizer, which brings together all the best of some other already good optimizers as for example the RMSProp. It additionally uses the idea of momentum optimization. [16, p. 302]
Since the Adam optimizer is a combination of the best features of different optimizers it is not surprising that it provided the best results as we can see in figure **??**

Figure 4.8: Test results - optimizers

Batch Normalization:
As already discussed we use a fully connected neural network consisting of an input layer, multiple hidden layers and an output layer. The hidden layers were only Dense layers put into a sequence after each other. To apply Batch Normalization to our model we compared the training and testing results to our model without the Batch Normalization. We applied the Batch Normalization by inserting a Batch Normalization layer after every Dense layer.
As we can see in figure **??** the Batch Normalization reduces our MAE significantly and was one of the most influential adjustments to our model so far.

Figure 4.9: Test results - Batch normalization

Dropout:
This adjustment can be classified by the same category as the former Batch Normalization. It was applied in the same way, by adding additional Dropout layers to the model after every Batch Normalization layer.
The results, as we can see in figure **??** showed, that ...

Figure 4.10: Test results - Dropout

Batch size:
16 - 256

Figure 4.12: Caption

Model architecture:
This adjustment is one of the biggest black boxes of all the adjustment we made. We spent a lot of time on testing different numbers of layers and neurons to end up with the following, best working solution.
As shown in figure 4.11, there is a tendency to a architecture, relating its number of neuron, that is descending.
As the best number of layers we found a architecture of 4-5 layers regarding the resulting MAE.

Figure 4.11: Test results -model architecture

After figuring out the best number of hidden layers and the ratio of the number of neurons for each layer, we tried additionally different values of neuron numbers. We can see in figure 4.12 that the best results were obtained by XXX-XXX-XXX-XXX.
Different feature sets:
At a last point, that counts not as classic hyperparameter, but ranks among training adjustments, we were testing different sets of features. Due to the fact, that testing every combination of all 55 features we were respecting is not pracitcable without having an enormous amount of computing power, we constrained us to manually picked sets of features.
Even though it is one part of the job of the neural network to find out which features have a big impact on the resulting value and which features decisive and related to the target value, it might help the neural network to learn quicker or more efficient if we sort out some of the irrelevant features before it even comes to training.
??

Figure 4.13: Test results - different feature sets

Chapter 5

Results of the different approaches

As already introduced we applied four different concepts to solve the given regression problem. We implemented the first three models by the sci-kit software library for python. The fourth model, our main approach, was implemented using the open-source library Keras.
As mentioned before we focused the most on the deep learning approach, regarding optimizing its results by determining the best parameters for the underlying regression problem.
Our main comparison criteria was the MAE of the resulting predictions. Secondary we took into account how well the models performed on specific groups of players, for example outliers with very high values.
This chapter only explains the experimental setup and provides and presents the results of them. The analysis and interpretation will happen in the consecutive chapter

5.1 Linear Regression

For the beginning we pay attention to the most simple of our four approaches.
On the contrary to the other three approaches we simply used the given class *LinearRegression* from sklearn.linear_model to apply linear regression to our regression problem.
Below we can see the results in figure **??**.

MAE:	114.742
RMSE:	419.346
MAE top 10%:	865.393
MAE bottom 90%:	145.498
Highest positive error:	5.532.167
Highest negative error:	-1.635.248

5.2 SVR

At first we had to think about the allocation of the hyperparameters of the SVM. We focused on the hyperparameters C, ϵ and γ. [19]
We used the class *SVR* from sklearn.SVM. To determine the best combination of those three different parameters we used another function of sklearn.model_selection, the class *GridSearchCV*. This class enabled us to obtain the best parameter combination automatically by passing a python dictionary containing all the to be tested values.
We decided to use the following range of values by trying out different combinations manually at first:
Values for:
Cs = [1e-3, 1e-2, 1e-1, 1]
epsilons = [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
gammas = [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 'scale', 'auto']

Where *scale* and *auto* are predefined options for the parameter *gamma*.
The result of the automated parameter tests have been the combination in figure **??**:

C:	1e-3
ϵ:	1e-8
γ:	1e-6

Allocating the SVM with this parameter allocation we obtained the prediction results of subsequently placed figure **??**:

MAE:	93.447
RMSE:	389.546
MAE top 10%:	498.393
MAE bottom 90%:	65.498
Highest positive error:	3.335.298
Highest negative error:	-1.136.279

5.3 Random Forrest

As already mentioned in section **??** the random forest model is not very sensitive to changes in its hyperparameters. We used the default parameterized class *RandomForestRegressor* of sklearn-ensemble to obtain the random forest prediction for our data.
Figure 5.1 shows the so far best result, as the MAE drops for the first time underneath the line of 80.000.

Figure 5.1: Results of the random forest approach

5.4 Deep Learning

Figure 5.2: Results of the deep learning approach

Approximation
Approximation good for low priced players.
But bad for highly priced players.
Total average error is about 70.000 - 100.000.
Around 300.000 for players with an actual price of 30.000 and higher.
Around 11.000 for players with an actual price of 30.000 and lower.

Chapter 6

Conclusion

List of Figures

Bibliography

[1] Ultimate team made up 28% of ea revenue last year. https://www.pcgamesn.com/fifa-ultimate-team-revenue. Accessed: 2020-04-18.

[2] Ea's ultimate team now worth $800 million annually. https://www.gamesindustry.biz/articles/2017-03-01-eas-ultimate-team-now-worth-USD800-million-annually. Accessed: 2020-04-18.

[3] Fifa 20 chemistry guide for ultimate team. https://www.ea.com/de-de/games/fifa/fifa-20/news/fifa-20-10-million-players-infographic. Accessed: 2020-01-05.

[4] Beginners introduction guide to fifa 20 ultimate team. https://www.fifauteam.com/beginners-introduction-fifa-20-ultimate-team/. Accessed: 2020-01-06.

[5] Fifa 20 chemistry guide for ultimate team. https://www.fifauteam.com/fifa-20-chemistry-guid/. Accessed: 2020-01-05.

[6] Fifa20 price ranges. https://www.fifauteam.com/fifa-19-price-ranges-ultimate-team/. Accessed: 2020-05-26.

[7] Fifa20 taxes. https://guides.gamepressure.com/fifa-20/guide.asp?ID=51784/. Accessed: 2020-05-26.

[8] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning* -. MIT Press, Cambridge, 2012.

[9] Ameet Talwalkar Mehryar Mohri, Afshin Rostamizadeh. *Foundations of Machine Learning*. Massachusetts Institute of Technology, Cambridge, Massachusetts, 2012.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Mariette Avad and Rahul Khanna. *Efficient Learning Machines*. ApressOpen, Cambridge, Massachusetts, 2015.

[12] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. *Journal of Machine Learning Research*, 12:199–222, 2004.

[13] Edward A. Bender. *Mathematical Methods in Artificial Intelligence*. IEEE Computer Society Press, Washington, DC, USA, 1996.

[14] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002. `https://www.researchgate.net/profile/Andy_Liaw/publication/228451484_Classification_and_Regression_by_RandomForest/links/53fb24cc0cf20a45497047ab/Classification-and-Regression-by-RandomForest.pdf`.

[15] W. J. Zhang, G. Yang, Y. Lin, C. Ji, and M. M. Gupta. On definition of deep learning. In *2018 World Automation Congress (WAC)*, pages 1–5, 2018.

[16] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems*. "O'Reilly Media, Inc.", Sebastopol, 2017.

[17] Futbin database. `https://www.futbin.com/20/players`. Accessed: 2020-04-06.

[18] " Gal Oshri". Introducing tensorboard.dev: a new way to share your ml experiment results. `https://blog.tensorflow.org/2019/12/introducing-tensorboarddev-new-way-to.html`, 2019. Accessed: 2020-06-06.

[19] Yuan Ren and Guangchen Bai. Determination of optimal svm parameters by using ga/pso. *JCP*, 5(8):1160–1168, 2010.