Predicting the Outcome of Soccer Matches

for the

Master of Science

from the Course of Studies Informationssysteme

at the Technische Hochschule in Ulm

by

Lisa Boos, Hemlata Prajapati, Khaled Jallouli, Till Hoffmann

July 2020

First Reviewer	Prof. Dr. Goldstein
Second Reviewer	Prof. Dr. Herbort

Author's declaration

Hereby I solemnly declare:

1. that this document, titled *Predicting the Outcome of Soccer Matches* is entirely the product of my own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;
2. I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;
3. this document has not been submitted either in whole or part, for a degree at this or any other university or institution;
4. I have not published this document in the past.

I am aware that a dishonest declaration will entail legal consequences.

Ulm, July 2020

Lisa Boos, Hemlata Prajapati, Khaled Jallouli, Till Hoffmann

Abstract

These days Machine Learning, Neural Networks or Data Mining are common buzzwords in the world of computer science. But a lot of people do not know what they are actually talking about, when using these kind of words. Therefore it makes sense to learn about neural networks and machine learning in general during the course of a masters degree in information technology. Fortunately there are many data sets online to use for analysis to get a basic understanding of the topic at hand. The European Soccer Database from Kaggle [6] is such a data set. During the team project of the information systems master at the Technische Hochschule Ulm it was our task to use the european soccer database for analytics in order to be able to predict the outcome of soccer matches. From the database we extracted several different features, like amount of goals shot by each team, amount of wins, draws and losses for each team as well as shot-accuracy and shot-efficiency per team and the overall ball possession of each soccer-team per match. Using an algorithm which we adapted for our special data [2] we created a sliding window which aggregates the extracted features over the last 10 soccer-games for each match in the database in chronological order. Because a lot of data samples in the database were incomplete regarding some of the extracted features, we created 5 different versions of the sliding window. Each containing a different amount of features and therefore a different amount of data samples. This gives us the possibility to test classifiers with various different models and later pick the solution that works best. The sliding window option 1 uses the least amount of features (13 features) and has the highest number of data samples (20823 data samples). Option 2 has 21 features and 7033 data samples, option 3 uses 29 features and 7033 samples, option 4 contains 25 features and 6996 samples and option 5 has the most features (33) and 6996 data samples. We used the 5 sliding window options to train and test various classifiers such as a Decision Tree, a Multi-Layer Perceptron and various different basic sequential neural nets. For the Decision Tree and the Multi-Layer Perceptron we used the scikit-learn API. For the basic sequential neural nets we used the Tensorflow/Keras framework, which is state of the art when doing data analytics tasks. The decision tree using the first sliding window option and a depth of 4 gives us a testaccuracy of 52,95%. The Multi-Layer Perceptron uses the sliding window option 1 aswell and has a test-accuracy of 53,45%. It has 2 hidden layers with 52 and 32 neurons. The best version of the Keras Sequential Neural Network was trained with sliding window option 3, has 2 hidden layers (21, 21 neurons) and a test-accuracy of 56,25%. It is also the best classifier in general we were able to find until now. At first sight model accuracies barely over 50% don't seem very good, but considering that the home team has a base chance of 46% to win the match and soccer is still a gambling game up to some point those accuracies aren't too bad at

all. Nevertheless, the team will continue its work on the project striving for better models by using additional features and varying the amount of features, adapting the split of training- and test-data, trying other kinds of classifiers and tweaking the existing models by changing parameters and input functions.

Contents

Acronyms

List of Figures

List of Tables

Listings

1. Introduction

The project is about predicting the outcome of soccer games and training a neural network. For the training the database from Kaggle in source [6] has been used.

1.1. Motivation

Machine Learning and Neural Networks is a main topic in the software development field. Many companies start to try analyze different kind of data with this approach. Because of this reason as master students in information systems it is very interesting to gather knowledge about machine learning and neural networks. For this matter a prediction of soccer matches is a very nice procedure to do. For this topic there is not many additional knowledge to gather, what you need for analyzing the data and prepare them for prediction. Additionally there is a free database with data of matches for a couple of years. In the beginning the knowledge about machine learning is very low and the goal is to improve the handling of python in combination with machine learning techniques. This includes pre-processing data with Pandas, normalizing data sets and extracting the right features, as well as developing adequate models for the machine-learning algorithm. Additionally to the gaining of knowledge we like to create an algorithm, which predicts the outcome of soccer games with a decent accuracy.

1.2. Goals

The main goals of this project can be divided into 2 parts:

- **Gaining knowledge**
  - Improve skills in python
    For the the project and for future work with neural networks it is necessary to become familiar with python. There are many things, which are different in python than in other programming languages. All these things have to be discovered and it is necessary to learn how to deal with the python syntax. Additionally its important to gain knowledge about some library, which you can use with python and make it easier to solve some problems.

  - Gaining knowledge about data pre-processing
    It is very important to edit the data in a way, that it is ready for a neural network. Algorithms can for example not deal with strings, only with numbers. For this matter, before there has to be some knowledge gained, that the data can be processed in a proper way.

  - Gaining knowledge about neural networks
    Neural networks is one of the most famous topics in the technical world these times. So as master students in computer science it is urgent to gain some knowledge about neural networks and machine learning. For the project it is necessary too to understand the overall technology of the whole topic.

- **Outcome of the project**

  - Finding the right features
    As a first step it should be done a feature selection. For this the database has to be analyzed and there should be choose some first features by gut feeling. This features has to be checked, if they are independent of each other. During the project it should be always reconsidered, if it make sense to add some more features and checked whether it improves the accuracy.

  - Normalizing the features in a proper way
    The features have to be normalized before using them for a prediction model. For this procedure it is necessary to find algorithms or write some. The normalization of the data is a major step in the project development.

  - Finding a good model for the prediction
    There are different libraries available for machine learning. The recommended library is Tensorflow in combination with Keras. Additional there has to be research for other libraries and approaches. The different models are compared and in the end the best model will be choose. It is important to look to the prediction of the actual winner than the prediction of the goals in a different way. So this step has to be done for both ideas.

  - Getting a decent accuracy with the prediction
    The accuracy for the winner should be higher than 50% in the end. With more than 50% you would theoretically always earning money, if you would bet everything, which the model is predicting. The main goal is to improve the accuracy during the whole project. For the predicting of the goals the accuracy can be lower than 50% because it is harder to predict the exact number of goals which were shot by each team.

– Creating a Homepage
  We want to provide a easy to use Homepage which makes it easy for people to look through our former predictions and predict upcoming soccer games. For this matter it is necessary to create a frontend for showing the data in a easy to understand and nice way, an API and a backend.

The first project team had the members Khaled Jallouli, Martin Schmidt, Sergej Dechant and Lisa Boos. The second Team Khaled Jallouli and Lisa Boos from the first team and the two new team members Hemlata Prajapati and Till Hoffmann. The knowledge gaining part was for both of the teams from major interest, except of the 'Gaining knowledge in data pre-processing', this part was only done by the first team. In the outcome of the project, 'Finding the right features', 'Normalizing the features' was only done by the old team. The 'Finding a good model for prediction' was done by the first team for the winner prediction and for the goal prediction by the new team. The quality criteria was a team work of only the new team as same as the 'Creating a Homepage' part. The team will change for the next step again, Hemlata Prajapati and Till Hoffmann will continue the project with maybe new students and Khaled Jallouli and Lisa Boos will leave the team.

1.3. Procedure

For the procedure process we decided to proceed in order to the CRISP-DM model. In image 1.1 you can see the different stages in order to CRISP-DM.

Figure 1.1.: The six stages of CRISP-DM [7]

- Business Understanding
  As first step we have to set the goals of the project which are already described in the chapter before. It is necessary to clear what exactly is the required outcome. For this matter in this project we use SCRUM to organize our project. We meet once per week to discuss the achievements from each single person. All 3-4 weeks there is a sprint meeting where we discuss the group achievements and if the project still leads to the right way. In the initial sprint meeting the common parameters of the project are set.

- Data Understanding
  Before we can start to create a model or select features we have to understand the Dataset. For this matter it is necessary to understand the structure of the set including the different columns and what they indicate. Additional it is important how many data the set includes and which types of value. It is important to learn also something about soccer and what are important facts about a game.

- Data Preparation
  Before it is possible to use the Dataset in a neuronal network it is necessary to prepare the data. For this matter as first step there has to be a feature selection. So one has to decide which columns are important for the later prediction. After we dropped the unnecessary columns it is important to prepare the resulting dataset. The records which have empty attributes have to be deleted or filled with specific values. Additionally the data has to be sorted and aggregated. At the end it has to be split into test and training data.

- Modeling
  The resulting dataset from the step before should be ready to be used for different models. In this step we will prepare different models to find a decent one. Additionally it is necessary to define a measurable goal how good the model can get.

- Evaluation
  In the end of this semester we will make a evaluation of the accomplished goals and if it is possible to increase the accuracy in the following semester.

- Deployment
  —-> TODO Till

The steps ‘Data Preperation’ and ‘Modeling’ will repeat as long as the resulting accuracy is not getting any better or we are satisfied with the resulting one. For all the steps it is necessary to do a lot of research to find the right techniques to do the tasks in a proper way. —-> TODO Till

2. Data Understanding

Data Profiling is the process of examining available data repeatedly throughout a project. At first light profiling assessment was undertaken. This was done using SQL and open source SQLite editors, entity relationship diagram tools and Pandas Profiling. As can be seen in the entity relationship diagram depicted in figure 2.1 the data available for this project is stored in a database consisting of seven tables.
The tables "Country", "League", "Team" and "Player" hold IDs, names and foreign keys to the tables "Team_Attributes", "Player_Attributes" and "Match". A player's birthdate, height and weight can be found in the table "Player".
The table "Team_Attributes" provides data on the individual teams such as playing style, defense class and so on in 12 numerical and 13 categorical columns or variables. However, 66.5 % of the values for "buildUpPlayDribbling" are missing and therefore this variable is useless.
The table "Player_Attributes" holds 31 numerical and 4 categorical variables and provides player statistics. Only a small percentage of values is missing.
The table "Match" holds information on football matches in Europe from 2008 until 2016 which is made up of 64 numerical and 19 categorical variables including goals scored, ball possession and odds quoted by several bookkeepers. Unfortunately a high percentage of the data on which players played in a match is missing. Moreover, ball possession and shot statistics are in XML format and not available for each match either. The odd variables (from different bookkeepers) are highly correlated.
Together with the product owners, it was decided that this project, or at least its first phase, and the features which have to be developed will be focusing on soccer matches and their outcomes and not on individual team- or player-attributes and statistics. Therefore, the tables "Team_Attributes" and "Player_Attributes" are not of interest in this phase. The tables "Country", "League", "Team" and "Player" are not relevant either, since they do not hold any useful information for the first phase.

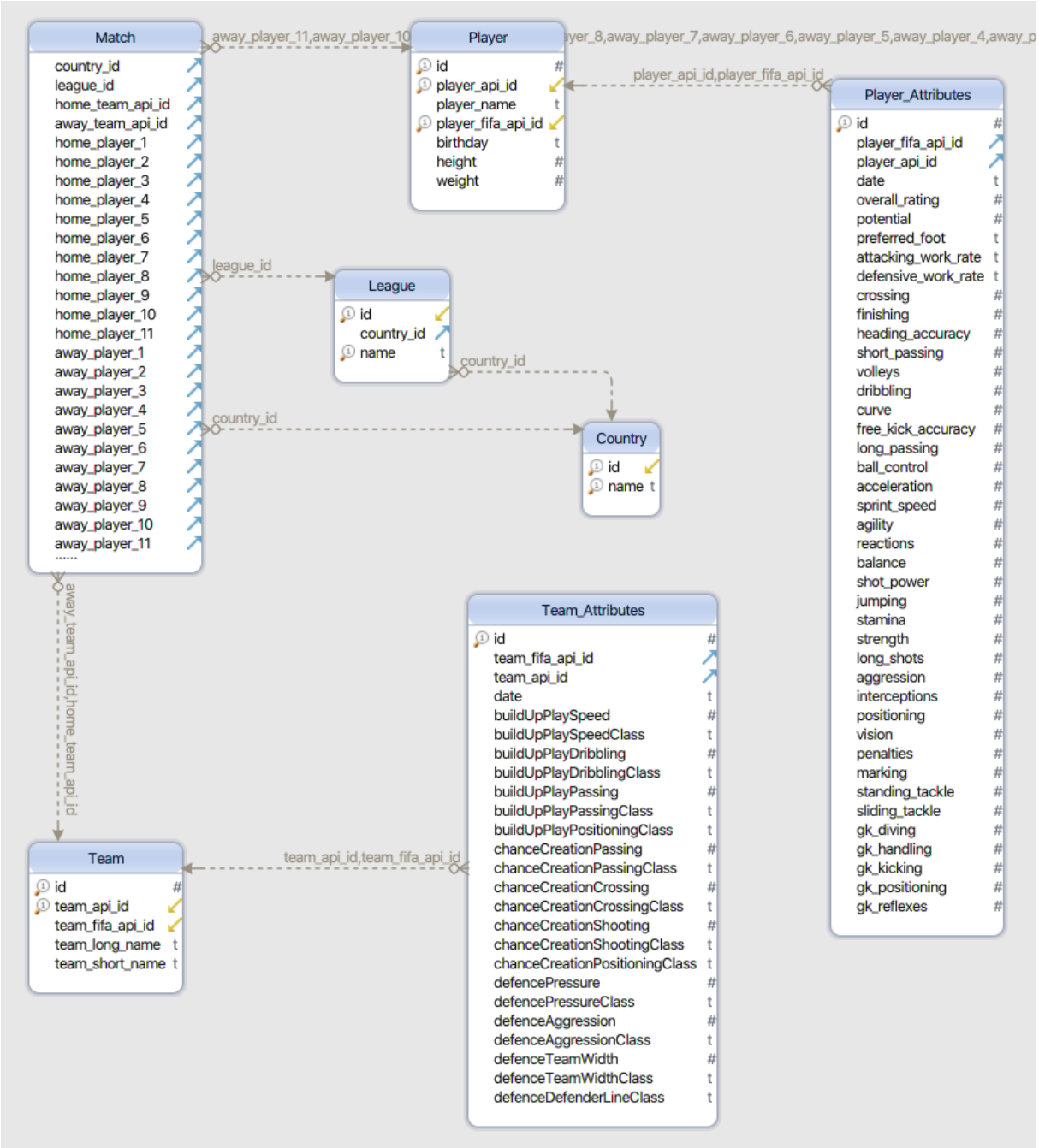

Figure 2.1.: Entity Relationship Diagram European Soccer Database

3. Feature Selection and Extraction

As stated in chapter 2 the preoject team agreed with the product owners, to focus on the features of soccer matches and their outcomes in this first phase of the project. Features like the number of scored goals per match/ team or the amount of won, lost or draw matches per team could be calculated directly out of the data at hand. Those features build the base dataset for the analyses we want to pursue. In detail, these are: Odds for the home-team winning, odds for a draw, odds for the away-team winning, amount of wins, draws and losses for the home-team, amount of wins, draws and losses for the away-team, amount of scored and received goals of the home-team and amount of scored and received goals for the away-team. In total there are 13 features in the base dataset. As said before, features about the ball possession or shot-statistics are stored as XML-documents within the database. After consultation with the product owners the team agreed to extract these values out of the XML-documents to get some additional features. The structure of these XML-documents looks like this:

```
<shoton>
  <value>
    <stats>
      <shoton>1</shoton>
    </stats>
    <event_incident_typefk>137</event_incident_typefk>
    <elapsed>8</elapsed>
    <subtype>distance</subtype>
    <player1>27462</player1>
    <sortorder>3</sortorder>
    <team>9912</team>
    <n>219</n>
    <type>shoton</type>
    <id>375900</id>
  </value>
</shoton>
<shotoff>
  <value>
    <stats>
      <shotoff>1</shotoff>
    </stats>
    <event_incident_typefk>9</event_incident_typefk>
    <elapsed>9</elapsed>
    <subtype>distance</subtype>
```

```
    <player1>30706</player1>
    <sortorder>1</sortorder>
    <team>8697</team>
    <n>220</n>
    <type>shotoff</type>
    <id>375902</id>
  </value>
</shotoff>
```

Listing 3.1: XML structure of shot-statistics

```
<possession>
  <value>
    <comment>51</comment>
    <event_incident_typefk>352</event_incident_typefk>
    <elapsed>23</elapsed>
    <subtype>possession</subtype>
    <sortorder>1</sortorder>
    <awaypos>49</awaypos>
    <homepos>51</homepos>
    <n>79</n>
    <type>special</type>
    <id>462628</id>
  </value>
</possession>
```

Listing 3.2: XML structure of ball-possession-statistics

The shoton and shotoff values can be aggregated per match and team to get features describing the total amount of shots and the amount of shots on the goal per team. Combined with the amount of scored goals per team additional features such as shot accuracy (ratio of shots on the goal to the total amount of shots) and shot efficiency (ratio of scored goals to the total amount of shots) can be calculated per team and match.
The additional features after the extraction are: home/away-shots, home/away-shots-on-target, home/away-shot-accuracy, home/away-shot-efficiency. The dataset counts 21 features in total now.
The values for ball-possession are given in percent for each half-time and if necessary for the overtime aswell. This way, the total amount of ball-possession in minutes per team can be summed up quite easily for the whole match.
After the extraction the following features are added to the base dataset: home-possession and away-possession. The features add up to 23 in total now.

4. Data Preprocessing

Based on [1] five sliding windows for match data were implemented. The idea of these sliding windows is to deliver match data in historical order and to aggregate data on a team's performance during the last 10 games. First the relevant data was extracted in chronological order with the following SQL statement:

```
SELECT id, country_id, league_id, season, date, home_team_api_id,
    away_team_api_id, home_team_goal, away_team_goal, B365H, B365D,
    B365A, shoton, shotoff, possession
FROM Match
ORDER BY date asc
```

Listing 4.1: SQL code for Sliding Window

Since the bookkeeper odds are highly correlated, as has been pointed out in chapter 2, only Bet365 data (B365H = odds home team wins, B365A = odds away team wins, B365D = odds draw) will be used.

4.1. Slliding Windows or Data Options

4.1.1. Option 1

For the first sliding window or option, data on shoton, shotoff and possession are not necessary and therefore are dropped. Moreover, rows without odds are dropped as well. The resulting data is saved as a CSV file:

```
conn = sqlite3.connect("../data/eusoccerdatabase.sqlite")
query = "SELECT id, country_id, league_id, season, date, match_api_id, 
    home_team_api_id, away_team_api_id, home_team_goal, away_team_goal, 
    B365H, B365D, B365A, shoton, shotoff, possession FROM Match ORDER BY 
    date asc"
df = pd.read_sql_query(query, conn)
df = df[np.isfinite(df['B365H'])] # drop rows without odds
df_noxml=df.drop(['shoton', 'shotoff', 'possession'], axis=1)
df_noxml.to_csv("../data/matches_all.csv")
```

Listing 4.2: Python code for matches_all.csv

This CSV file consists now of 22592 rows each representing one football match including information on how many goals each team scored, what the odds were and the date of the match:

date	match_api_id	home_team_api_id	away_team_api_id	home_team_goal	away_team_goal	B365H	B365D	B365A
2008-08-09 00:00:00	483129	8583	9830	2	1	2.10	3.10	3.75
2008-08-09 00:00:00	483130	9827	7819	2	1	1.57	3.60	6.50
2008-08-09 00:00:00	483131	9746	9831	1	0	2.30	3.00	3.40
2008-08-09 00:00:00	483132	8682	8689	0	1	2.10	3.10	3.80
2008-08-09 00:00:00	483134	9829	9847	1	0	2.40	3.10	3.10

Figure 4.1.: Data for Option 1

This CSV file was then used for the first sliding window. The code for it [2] expects a CSV file and processes it row by row. For every match the outcome of the game based on scored goals is calculated. Moreover, statistics on how each of the two opponents performed in the previous ten games are generated:

	result	odds-home	odds-draw	odds-away	home-wins	home-draws	home-losses	home-goals	home-opposition-goals	away-wins	away-draws	away-losses	away-goals	away-opposition-goals
0	H	1.36	4.50	9.00	5	3	2	14	10	2	2	6	8	14
1	D	2.25	3.00	3.50	3	4	3	16	12	4	3	3	14	12
2	D	2.80	3.00	2.70	5	2	3	10	8	4	4	2	15	13
3	A	1.91	3.25	4.33	5	3	2	18	11	3	3	4	10	11
4	H	2.10	3.00	4.00	2	4	4	6	9	1	6	3	7	10

Figure 4.2.: Sliding Window Option 1

As an example we can see in row 0 that the home team won, which is indicated by a capital "H". The odds that the home team wins were 1.36, 4.50 for a draw and odds of 9.0 that the away team wins. In the last 10 matches the home team won 5 times, finished 3 matches with a draw, lost 2 times and scored 14 goals. Opposing teams were able to score 10 goals. Subsequent columns contain the equivalent statistics for the opposing/away

team. The first sliding window reduces the total dataset to 20823 rows and generates 13 features. Furthermore, now 3 distinguishable classes (H = home team wins, A = away team wins, D = draw) are available.

4.1.2. Option 2

This option, which uses a slightly enhanced version [3] of the first sliding window code, adds statistics on goal shots:

home-shots	home-shots_on_target	home-opposition_shots	home-opposition_shots_on_target
137	67	117	53
134	64	151	77
120	58	124	56
177	82	74	37
161	72	74	31

Figure 4.3.: Sliding Window Option 2

Unfortunately due to the poor choice of data origin this option reduces the dataset to 7033 rows. But on the bright site, this option extends the number of features to 21.

4.1.3. Option 3

Based on option 2, option 3 additionally calculates shot accuracy statistics with code written and executed in a Jupyter Notebook [8]:

home_shot_accuracy	home_shot_efficiency	home_opposition_shot_accuracy	home_opposition_shot_efficiency
0.489051	0.164179	0.452991	0.301887
0.477612	0.125000	0.509934	0.207792
0.483333	0.172414	0.451613	0.267857
0.463277	0.268293	0.500000	0.324324
0.447205	0.208333	0.418919	0.258065

Figure 4.4.: Sliding Window Option 3

Option 3 has 29 features and 7033 rows. "x_shot_accuracy" represents the percentage of the shots which actually hit the goal or goal keeper and "x_shot_efficiency" the percentage of the shots which actually were counted as goals.

4.1.4. Option 4

Option 4 again uses an enhanced enhanced version [4] of the first sliding window code and extends option 2 with ball possession statistics:

home-possession	home-opposition_shots	home-opposition_shots_on_target	home-opposition_possession
481	117	53	429
540	69	36	369
477	74	31	433
439	133	77	471
435	153	73	475

Figure 4.5.: Sliding Window Option 4

Ball possession is provided in minutes. This option has 6996 rows and 25 features.

4.1.5. Option 5

The last option is based on option 3 and additionally includes ball possession statistics. It has the highest number of features (33) and 6996 rows. The Python code for it can be found in the same Jupyter Notebook as for option 3 [8].

4.2. Data Profiling Part 2

Using Pandas Profiling all sliding windows or options were profiled again. Here is and excerpt of the profile for option 3:

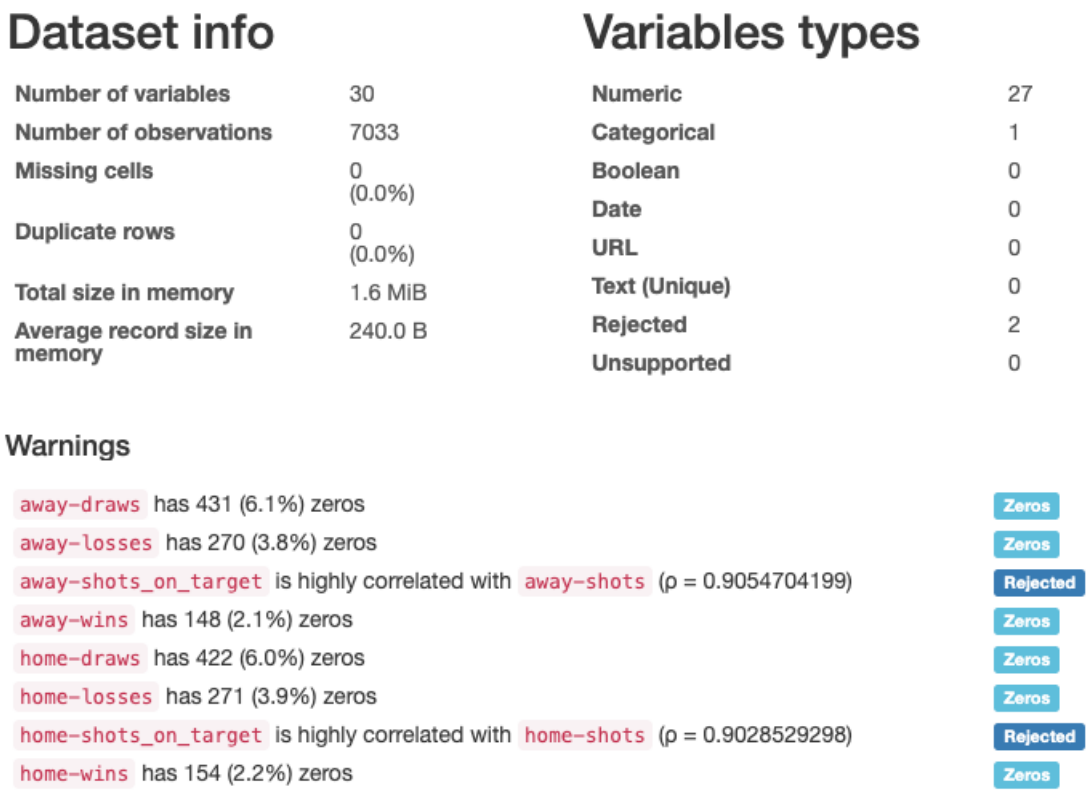

Figure 4.6.: Pandas Profiling Sliding Window 3

Apart from one variable, which holds the classes respectively the outcome H, A, or D of a football match, all features/columns are numerical and no values (cells) are missing. As expected x-shots and x-shots_on_target are highly correlated. Interestingly zero values are marked as warnings. However, in this context they are fine and still usable. Without having domain knowledge and purely relying on tools, one might be inclined to drop these values and lose valuable information. Another interesting information found in Pandas profiles is the total size in memory. 1.8 MiB shows that the data for option 3 is little and definitely small enough to fit into RAM or GPU memory. This information is valuable, as will be shown later.
Since the profiles for the other options are similar to the one described above, they will not be explained.

4.3. Normalization

Since the input data varies in scale it has to be normalized, except the "result" column which has to be encoded. Normalization is the process which normalizes all input data

avoiding negative effects in regards to decisions of a neuron. The following lines of code normalize the data in a dataframe:

```
from sklearn import preprocessing

column_names_to_not_normalize = ['result']
column_names_to_normalize = [x for x in list(dataframe) if x not in
    column_names_to_not_normalize ]
x = dataframe[column_names_to_normalize].values
x_scaled = preprocessing.normalize(x)
df_temp = pd.DataFrame(x_scaled, columns=column_names_to_normalize,
    index = dataframe.index)
dataframe[column_names_to_normalize] = df_temp
```

Listing 4.3: Python code for normalization

4.4. Encoding

The classes H,A,D respectively the outcome of football match which shall be predicted, have to be encoded before they can be used. Using sklearn this can be achieved with only a few lines of code:

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit([ "H", "A", "D"])
dataframe.loc[:,['result']]=le.transform(dataframe['result'])
```

Listing 4.4: Python code for encoding classes

4.5. Functions and Input Data For Neural Networks

The repetitive data preparation tasks encoding, normalization, splitting data into training and test data as well as getting data and prediction labels have been solved using Python functions. Furthermore, Pandas dataframes cannot be used as input for neural networks written with Tensorflow and Keras. The complete code, which has been partly described above, can be found here [5].
Since the data for all options is very little and fits completely into RAM, as has been described above, no input functions for Tensorflow models, e.g. neural networks, had to be written. An excerpt of how the final input data as numpy array for the neural networks described in chapter 5 looks like can be seen here:

odds-home	odds-draw	odds-away	home-wins	home-draws	home-losses	home-goals	home-opposition-goals
0.049957	0.165301	0.330601	0.183667	0.110200	0.073467	0.514268	0.367334
0.077897	0.103862	0.121172	0.103862	0.138483	0.103862	0.553931	0.415448
0.109311	0.117119	0.105407	0.195198	0.078079	0.117119	0.390396	0.312317
0.068789	0.117049	0.155945	0.180075	0.108045	0.072030	0.648271	0.396166
0.108097	0.154424	0.205899	0.102949	0.205899	0.205899	0.308848	0.463272

Figure 4.7.: Excerpt of Input Data for Neural Networks as Numpy Array

5. Modeling

In this phase, we have successfully completed the data preparation phase and our five sliding windows are ready to use.

Modeling is an iterative process, in which we can apply several modeling techniques to the same problem using the default parameters and then fine-tune them until we satisfy our quality criteria. There is not a single model and a single execution which can satisfactorily answer our questions. For this, we tested several models to find the one that best fits our problem.

This phase comprises tasks such as select modeling techniques, generate test design, build model and assess model.

5.1. Predicting the winner

5.1.1. Select Modeling Techniques

As a first step in modeling, we decided to choose Supervised Machine Learning algorithms, to perform multi-class classification because our objective is to predict the final result of a match between two teams, if there is a win by the home team, a draw or a win by the away team.

Therefore, we have selected Decision Trees and Neural Networks as techniques in order to test its performance and find the most appropriate for our project.

5.1.2. Generate Test Design

This part refers to the generation of a procedure to test the model quality and validity needs, before building our models.

For some modeling techniques, we have divided our dataset into training and test sets, the model is built based on the training set, and its quality is estimated based on the test set, which represents 30% of the dataset.

We also took care to not shuffle the dataset as we need to keep the last 10 matches of the sliding windows dataset in the correct order.
For others, we only used 5% for test set, a small amount because we wanted to keep as much data as possible for training and validation. The remaining 95% are divided into 80% training and 20% validation datasets.

For training the models, we used automated stop as a strategy, after the training loss did not improve more than 0.0001 for 10 consecutive epochs or the model exceeds 1000 training epochs.

To evaluate the models, we used the accuracy results as criteria.

5.1.3. Build Models

The aim of this part is to build several models before comparing the results.
Most modeling techniques have a number of parameters that can be adjusted to control the modeling process.

For our Supervised Machine Learning Algorithms, we used scikit-learn API to build a Decision Tree and Multi-Layer Perceptron neural network. We also used the TensorFlow/Keras framework to build basic sequential neural networks.

The Decision Tree can be modified by adjusting the depth of the tree. For Neural Networks, we can change the number of hidden layers, the neurons per layer and other parameters.

Decision Tree Classifier

A decision tree is a simple classification representation that learns from the data with a set of if-then-else decision rules.

Using the decision tree algorithm, we start at the root of the tree and divide the data on the feature that results in the largest information gain. We can then repeat this procedure until the leaves are pure.

In our project, we set the depth of the tree to four.
The Decision Tree Classifier achieved an accuracy of 52.95% using the first sliding window option as a dataset, as shown in the following Figure.

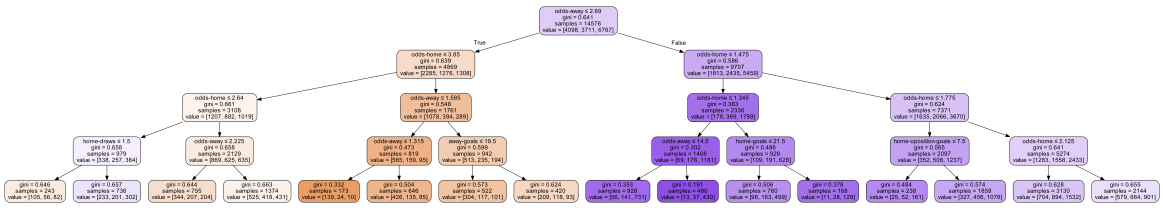

Figure 5.1.: Decision Tree (max_depth=4)

Multi-layer Perceptron

Multi-layer Perceptron (MLP) is a supervised learning algorithm consisting of three layers: one input layer, one hidden layer, and one output layer. The units in the hidden layer are fully connected to the input layer, and the output layer is fully connected to the hidden layer.

In our MLP Model, we used the first sliding window with 13 features in the input layer, two hidden layers, 52 neurons in the first one and 32 neurons in the second one. For the output layer, we have 3 neurons.

To be able to solve our problem, we used the sigmoid activation function(logistic) for the hidden layers and the softmax activation function for the output layer. We also used a stochastic gradient descent optimizer as a solver.
As we see in the following plot, the graph of the cost function indicating that the training algorithm converged after the 90th epoch.

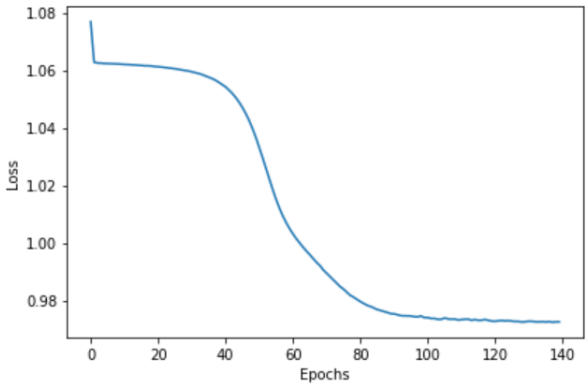

Figure 5.2.: MLP Cost function

The last step is to evaluate the performance of the model by calculating the accuracy of the prediction. We obtained 53.45% for the training dataset and 52.77% for the testing dataset.

Keras Sequential Neural Network

To build the model, we used Sequential as a model type. Sequential is the easiest way to create a model in Keras. It allows to build a model layer by layer. Each layer has weights that correspond to the layer that follows it.
The chosen layer type is 'dense'. Dense is a standard layer type that works for most cases. In a dense layer, all the nodes in the previous layer connect to the nodes in the current one.
As activation function for the hidden layers 'Rectified Linear Activation' (ReLU) was used and 'Softmax' for the output layer. Softmax sums the output up to 1 so that the output can be interpreted as probabilities. The model will then make its prediction according to the option which has a higher probability.

The first layer needs an input shape. The input shape specifies the number of rows and columns in the input.
The last layer is the output layer. It has three nodes - one for each option: Home Win, Draw or Away Win, which is for our prediction, as shown in the following line of codes.

```
model = tf.keras.Sequential([
  layers.Dense(13, activation='relu',input\_shape=(train\_X.shape[1],)),
  layers.Dense(16, activation='relu'),
  layers.Dense(8, activation='relu'),
  layers.Dense(3, activation='softmax')
])
```

Listing 5.1: Python code for simple Keras Sequantial Model Instantiation

To compile the model, we chose Adam as an optimizer. The Adam Optimizer adjusts the learning rate throughout the training.
The learning rate determines the speed at which the optimal weights for the model are calculated.
For the loss function, we chose *sparse_categorical_crossentropy*. It is one of the most common choices for classification. A lower score indicates that the model is performing better.

Weight regularization is a regularization technique that provides an approach to reduce over-fitting of a deep learning neural network model on training data and to improve the performance of the model on new data.
By default, no regularizer is used in layers. For this, we made some models with the addition of the L2 regularization, which is the sum of the squared weights.

In other models, we have added dropout, which is another regularization technique for neural networks, to avoid over-fitting in neural networks. Additionally we combined both regularization techniques in one model. Table 5.1 shows the test-accuracies of the models in comparison.

The evolution of the loss-function of the generated models over time can be seen in Figure 5.3.

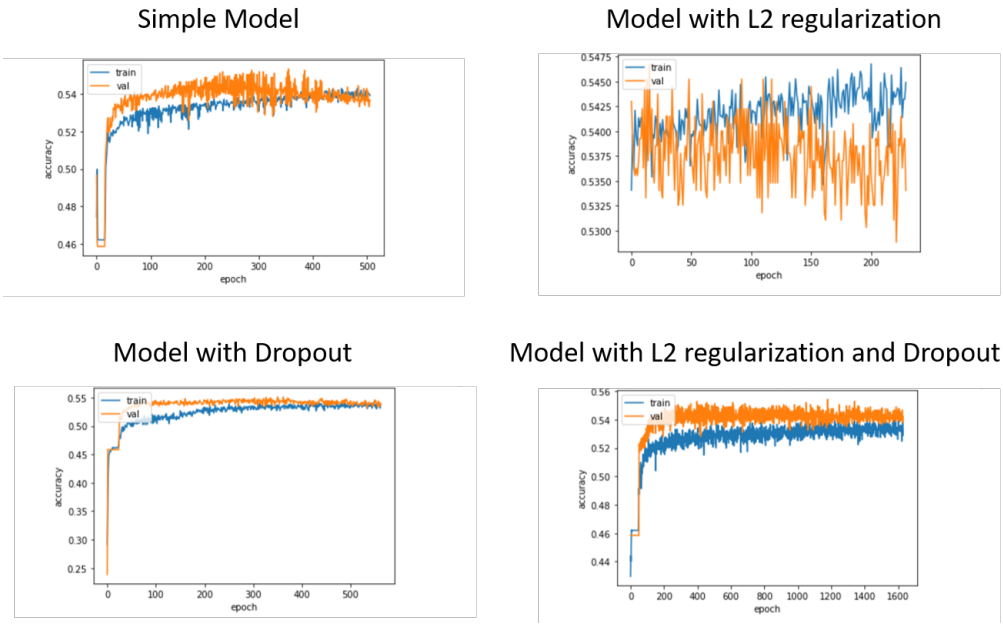

Figure 5.3.: Evolution of the loss function of the models over time

To get a better feeling for the impact the number of hidden layers and the amount of neurons within each hidden layer have on the overall performance of the model, we conducted a series of tests.
We tested each sliding window option with a varying amount of hidden layers (**H1**-**H4**) and a varying number of neurons per layer (**H**igh, **M**edium, **L**ow and **F**unnel).
The architectures of the neural nets can be found in section A.1.

	Normal	L2-Weight Regularization	Dropout	Dropout & Regularization
Model01	0.5220729	0.5191939	0.5191939	0.5268714
Model02	0.5198864	0.52272725	0.5369318	0.50852275
Model03	0.4943182	0.4971591	0.5113636	0.50852275
Model04	0.5057143	0.5114286	0.49142858	0.4857143
Model05	0.5	0.49714285	0.4942857	0.50285715

Table 5.1.: Test-accuracy of various models with different parameters

Model01	H	M	L	F
H1	0.537428	0.53358924	0.5460653	-
H2	0.53262955	0.537428	0.53454894	-
H3	0.5422265	0.537428	0.47408828	0.5393474
H4	0.5431862	0.5393474	0.537428	0.537428

Table 5.2.: Test-accuracies for variation of hidden layers and neurons for sliding window option 1

The test-accuracies for the models based on sliding window option 1 can be seen in Table 5.2.

Table 5.3 shows the test-accuracies for the models based on sliding window option 2.

Table 5.4 shows the test-accuracies for the models based on sliding window option 3.

Table 5.5 shows the test-accuracies for the models based on sliding window option 4.

Table 5.6 shows the test-accuracies for the models based on sliding window option 5.

Model02	H	M	L	F
H1	0.52840906	0.53409094	0.52840906	-
H2	0.5198864	0.5255682	0.53125	-
H3	0.5255682	0.53125	0.5369318	0.5369318
H4	0.5198864	0.5198864	0.53977275	0.5255682

Table 5.3.: Test-accuracies for variation of hidden layers and neurons for sliding window option 2

Model03	**H**	**M**	**L**	**F**
H1	0.53977275	0.5426136	0.5511364	-
H2	0.5625	0.5568182	0.54545456	-
H3	0.53977275	0.5369318	0.5625	0.53977275
H4	0.5625	0.5426136	0.54545456	0.5426136

Table 5.4.: Test-accuracies for variation of hidden layers and neurons for sliding window option 3

Model04	**H**	**M**	**L**	**F**
H1	0.5342857	0.5342857	0.5342857	-
H2	0.5257143	0.5371429	0.5342857	-
H3	0.5314286	0.5342857	0.5314286	0.52
H4	0.5285714	0.5285714	0.5285714	0.5228571

Table 5.5.: Test-accuracies for variation of hidden layers and neurons for sliding window option 4

Model05	**H**	**M**	**L**	**F**
H1	0.5342857	0.52	0.5314286	-
H2	0.5228571	0.5314286	0.5371429	-
H3	0.5342857	0.5228571	0.5314286	0.5371429
H4	0.5228571	0.5257143	0.5314286	0.52

Table 5.6.: Test-accuracies for variation of hidden layers and neurons for sliding window option 5

5.2. Predicting the final goals

5.2.1. Regression

Regression is used in a problem when the output variable is a real or continuous value, such as "number of goals" in our project. We are using machine learning regression algorithms i.e. multi-output regressor, decision tree regressor, random forest regressor, mlp regressors and Deep Learning multi perceptron neural networks with Keras to perform regression in our project to predict the number of goals scored per each team. Keras is a deep learning library that wraps the efficient numerical libraries Theano and TensorFlow. Main steps involved in the regression are-load a CSV dataset and make it available to Keras, preprocessing, create a neural network models with Keras for a regression problem, apply quality criteria to the models, tune the network topology of models with Keras and selection of the best model among all for making predictions of new data.

5.2.2. Data Preprocessing part

We have first applied the common data preprocessing to our dataset Sliding02goals to normalize the data values.
The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. The normalization is restricted to the input features and not output features.
Secondly, we have done data encoding for all the goals in the range of -1 to 1 where -1 means no goal and 1 is for 10 or more goals. The following figure Figure 5.4 describes the encoding done on output data values in preprocessing

```
In [ ]: def encodelarger(i):
            switcher = {
                0: -1,
                1: -0.8,
                2: -0.6,
                3: -0.4,
                4: -0.2,
                5: 0,
                6: 0.2,
                7: 0.4,
                8: 0.6,
                9: 0.8,
                10: 1,
            }
            # 1 be assigned as default value of passed argument (if goals > 10)
            return switcher.get(i, 1)
```

Figure 5.4.: Data Preprocessing for regression

5.2.3. Build Models

First, we have five different machine learning regression algorithms to build models and then compared their results in term of test accuracy in the evaluation part.

We have also used plain multiperceptron model as deep learning artificial neural network algorithm with keras . We can create Keras models and evaluate them with scikit-learn because scikit-learn is good at evaluating models and allow us to use powerful data preparation and model evaluation schemes with very few lines of code.

Multi Output Regressors

First one is Multioutput regression which consists of fitting one regressor per target. This is a simple strategy for extending regressors that do not natively support multi-target regression. Multioutput regressor can be created using Gradient Boosting Regressor and support vector machines to make predictions on test data.
Gradient Booster works on boosting or improving the week learner predictions and involve a loss function to be optimized, a week learner to make predictions and an additive model to add weak learners to minimize the loss function. It builds an additive model in a forward stage-wise fashion; it allows for the optimization of random distinguishable loss functions. We have kept random state parameter as zero in our model to avoid change in the random seed given to each Tree estimator at each boosting iteration.
Second multioutput regressor used is support vector regressor which uses the same principle as for classification and it relies on kernel functions. We have test Support Vector Regression (SVR) for a regression problem with two outputs. This means that Y tarin data has two values for each sample. Since SVR can only produce a single output, we have used the MultiOutputRegressor from Scikit.

5.2.4. Decision Tree Regressor

The decision trees is used to predict simultaneously the noisy x and y observations of a circle given a single underlying feature. As a result, it learns local linear regressions approximating the circle. Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.In our model we have used random state as 0,max depth equals to 1 and mean square error as criterion.

5.2.5. Random Forest Regressor

A random forest is an ensemble model that consists of many decision trees. Predictions are made by averaging the predictions of each decision tree. It is a forest that is a collection of trees. This makes random forests a strong modeling technique that's much more powerful than a single decision tree. Also, Random forest is a bagging technique and not a boosting technique. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees. We have krpt random state as zero while fitting model to the training data.

5.2.6. MLP Regressor

It is Multi-layer Perceptron regressor. This model optimizes the squared-loss using Adam or stochastic gradient descent. Both optimizers are used to update weights for better predictions. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. We have created MLP model two times using two different optimizers which are Adam and (Stochastic Gradient Descent)SGD.

Results using Adam Optimizer

Adam is an optimization algorithm that can be used to update network weights iterative based in training data. We have used three hidden layers of size 100 units, 30 units and 11 units respectively. Adam provides combined benefits of Adaptive Gradient Algorithm and Root Mean Square Propagation. Rather than using the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam use the average of the second moments of the gradients (the uncentered variance).

Results using Stochastic Gradient Descent Optimizer

Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. It is a classical optimization algorithm. We have used three hidden layers of size 50 units, 20

units and 11 units respectively. Activation is kept as relu and learning rate as constant.

5.2.7. Regression using Keras Deep Learning

This part describes the usage of deep learning techniques and keras deep learning library that wraps the efficient numerical libraries Theano and TensorFlow to do regression or we can say to predict the number of goals for each teams. It includes the following common main steps- Import the dataset, preprocessing of dataset, baseline model building, making predictions, evaluate models on the basis of quality criteria and parameter tuning.

Data Preprocessing

In data preprocessing section, normalisation has performed on input features data values and data encoding has performed.But, now there is a little change in the encoding section where we have encoded -1 as no goal and 1 as 5 or more goals rather than 10 or more goals.
The following figure Figure 5.5 describes the encoding done on output data values in preprocessing step where 0 goal encoded to -1, 1 goal encoded to -0.6, 2 goals encoded to -0.2, 3 goals encoded to 0.2, 4 goals encoded to 0.6, and 5 or more goals encoded to 1.

In [8]:
```
def encode(i):
    switcher = {
        0: -1,
        1: -0.6,
        2: -0.2,
        3: 0.2,
        4: 0.6,
        5: 1,
    }
    # 1 be assigned as default value of passed argument (if goals > 5)
    return switcher.get(i, 1)

def decode(i):
    switcher = {
        -1: 0,
        -0.6: 1,
        -0.2: 2,
        0.2: 3,
        0.6: 4,
        1: 5,
    }
    return switcher.get(i, "ERROR! Use Encode Before!")
```

Figure 5.5.: Updated Data Preprocessing for regression

Develop a Baseline Neural Network Model

In this section we will create a baseline neural network model for the regression problem. We can create Keras models and evaluate them with scikit-learn which allow us to excels at evaluating models and will allow us to use powerful data preparation and model evaluation schemes with very few lines of code.
Below we define the function to create the baseline model to be evaluated. It is a simple model that has a densely connected hidden layer with the higher number of neurons as input attributes (30). The network uses good practices such as the rectifier activation function for the hidden layer. No activation function is used for the output layers because it is a regression problem and we are interested in predicting numerical values directly without transform.
Aso, the mean squared error and mean absolute errors are our loss functions which is an estimate of how accurate the neural network is in predicting the test data. We can also used 20 percent of training data for validation and 80% of the training data is used to test the model, while the remaining 20% is used for testing purposes.
The baseline artificial neural network model for regression in shown in the following figure Figure 5.6

```
def build_model():
  model= tf.keras.models.Sequential()
  model.add(tf.keras.layers.Dense(units=30, activation='relu', input_shape=(train_X02.shape[1],)))
  model.add(tf.keras.layers.Dense(units=20, activation='relu'))
  model.add(tf.keras.layers.Dense(units=10, activation='relu'))
  model.add(tf.keras.layers.Dense(units=2))

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse','accuracy'])
  return model
```

Figure 5.6.: Baseline model for regression using deep learning

Train Model

The model has trained for 1000 epochs by keeping validation split of 0.2 and the training and validation accuracy is recorded in the history object. we have observed that the more epochs are run, the lower our MSE and MAE become, indicating improvement in accuracy across each iteration of our model.
keras is calculating both the training loss and validation loss, i.e. the deviation between the

predicted y and actual y as measured by the mean squared error. Let's see our respective losses plot using graph which compare the validation loss and training loss in following figure Figure 5.7

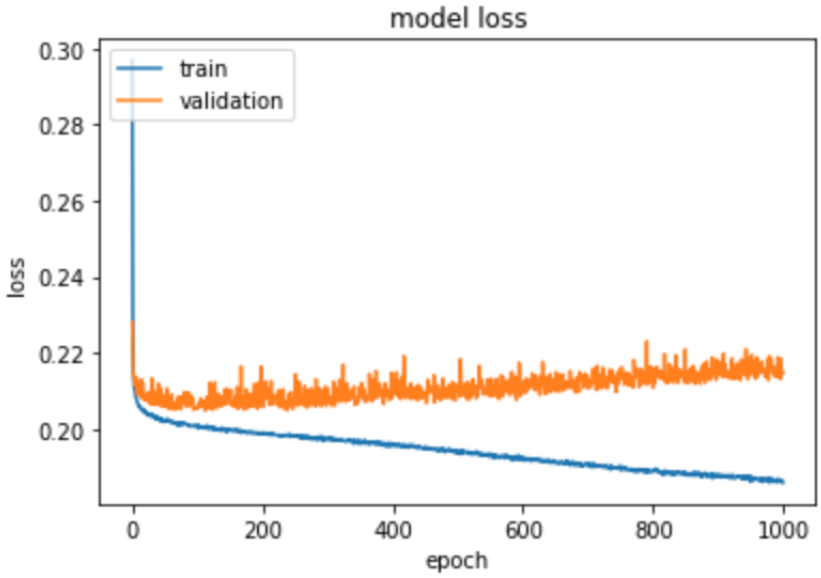

Figure 5.7.: Display of Validation and Training Loss

Make Predictions and Evaluate Models using Quality Criteria

We have predicted the number of goals using data in the testing set, however the predictions we have received are in the encoded form which we have decoded using our decode functions. The predictions made on test data are shown in the following figure Figure 5.8

-	QM-HG-Train	QM-AG-Train	FinalQM-Train	QM-HG-Test	QM-AG-Test	FinalQM-Test
Model	0.83	0.84	0.83	0.81	0.83	0.82

Table 5.7.: Quality Model for Away Team and Home Team

```
In [35]: y_train_pred

Out[35]: array([[-0.49964523, -0.48719516],
                [-0.4722548 , -0.6432786 ],
                [-0.48331586, -0.5827433 ],
                ...,
                [-0.43938887, -0.5005152 ],
                [-0.53124684, -0.49210525],
                [-0.14386953, -0.6515794 ]], dtype=float32)

In [36]: train_y02

Out[36]: array([[-0.2, -0.6],
                [-0.2, -0.2],
                [-0.6, -0.2],
                ...,
                [-0.2,  0.6],
                [-1. ,  0.2],
                [-0.2, -1. ]])
```

Figure 5.8.: Predictions on Test Dataset

Finally, the evaluation of models has been performed using Quality criteria explained in the Chapter 6 quality criteria section of the report. Then we calculate the final quality model for both teams away team and home team respectively. The results of the quality models are shown in the following table where QM is abbreviation for Quality Model, HG is abbreviation for HomeTeam Goals and AG is abbreviation for AwayTeam Goals

5.2.8. Multi Class Classification

Another idea for predicting the final goals is to use multi class classification. For this matter a common classification model can be used.

```
inputs = tf.keras.layers.Input(shape=(21,))
layer1 = layers.Dense(10, activation="relu", name="layer1")
layer2 = layers.Dense(20, activation="relu", name="layer2")
layer3 = layers.Dense(6, activation="softmax", name="layer3")
model = layer3(layer2(layer1(inputs)))

home-team_1 = model
```

```
8  away-team_2 = model
9  model = tf.keras.models.Model(
10    inputs=inputs, outputs=[home-team_1, away-team_2])
11 model.compile(optimizer="Adam", loss='sparse_categorical_crossentropy',
       metrics=[ "acc"])
12 history = model.fit(x, (y_home-team, y_away-team), epochs=100)
```

Listing 5.2: Python code for multi class classification

As the reader can see in listing 5.2 the used model has one input layer with the shape of 21 neurons, because there are 21 features. There are 2 hidden layers with 10 and 20 neurons and one output layer with 6 neurons. The 6 neurons are because we are only predicting goals from 0 to 5 and if the goals are greater than 5 (which does not happen very often, as it is described in the Data Preprocessing part) we count them as 5 goals. In line 5 it is shown how the model is created from the defined layers before. This model is used for both predictions (line 7 and 8), the home team score prediction and the away team score prediction. In line 11 is the model compilation with the two output vectors and in line 12 the training with these vectors. So instead of applying the model on only one output vector it is just used for two of them.

```
1 model.evaluate(test02,( testLabels_hometeam, testLabels_awayTeam))
```

Listing 5.3: Python code for multi class classification model evaluation

As the reader can see in listing 5.3 the evaluation has to be adjusted in the same way. It is necessary to provide the model with two evaluation vectors as well. The result is a array with 5 values, the whole loss of the model, the loss per team and the accuracy per team. With this model we achieve an accuracy of 30.53% for the home team and 37.64% for the away team.

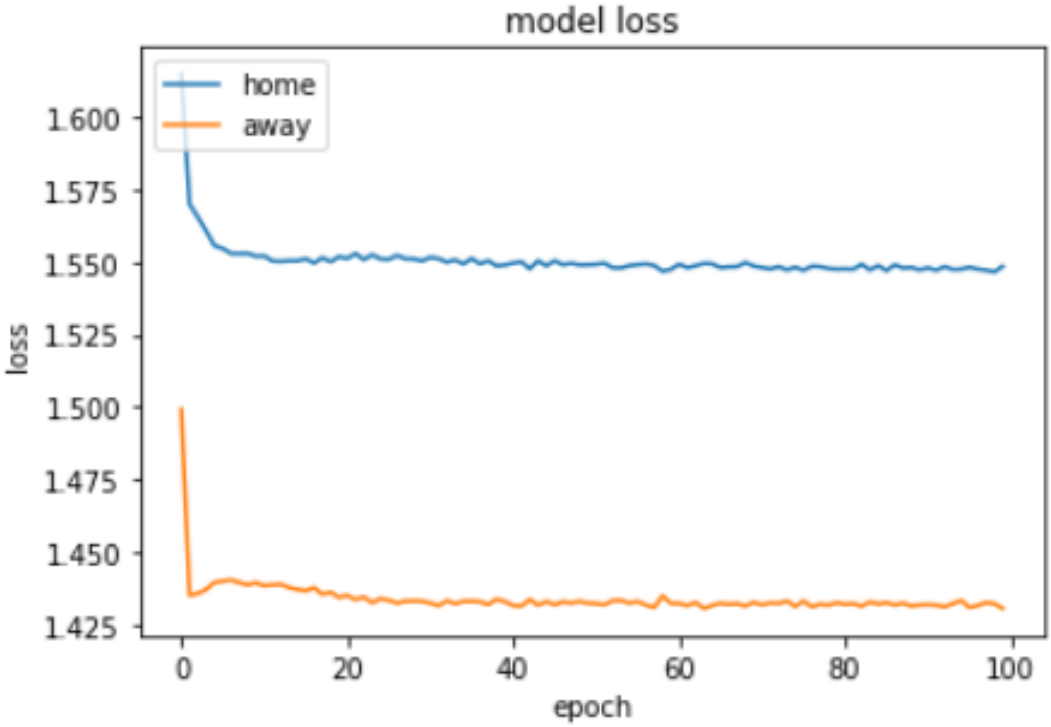

Figure 5.9.: Multi Class Classification loss graph

In figure 5.9 the reader is able to see, that the loss is higher for the home team than the away team. In the beginning the value is decreasing very much and after approximative epoch 10 the value is quite stable. The same shows the accuracy graph in figure 5.10. The accuracy for the away team is better than the one for the home team. And it is only increasing in the beginning. Afterwards it is a little bit noisy but the accuracy is not increasing significant in any epoch.

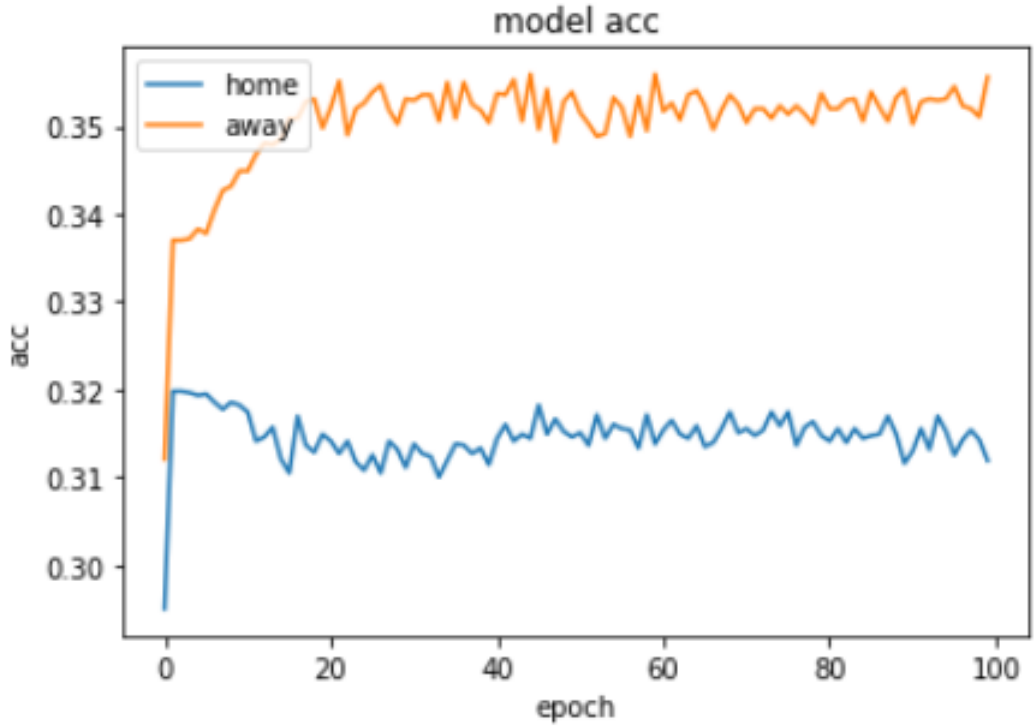

Figure 5.10.: Multi Class Classification accuracy graph

After using the own built quality criteria this approach reaches 80% accuracy. In this point there was no comparison with different models done. This will happen in the next semester. Maybe more hidden layers or more neurons can improve the outcome. Another idea for improving the result is to try the different sliding windows approaches. For this test sliding window 02 was used.

6. Evaluation

6.1. Predicting the winner

In this phase of the CRISP-DM Process we are going to evaluate the models and the features, we previously generated.
The evaluation criteria for the models and in the same way for the features which are used, is based on the test-accuracy. The highest accuracy we reached jet, have been 56.25% with the Sliding Window Option 1, which means with the highest amount of features. The more features we have the less training samples are there for our training, so it could be better to train with more samples and less features. But because of the reason that every models are in the same range, which means there is no model which has a much smaller accuracy, it is not possible to tell the one with the highest accuracy the best model or has the best feature selection in every case. If you are using different data or a higher or smaller amount of features it could be possible to reach a better accuracy with another model than the one which reached the highest accuracy in the actual case. The first ranked model is using Keras Sequantial Neural Network, as shown in Table 6.1:

Rank	**Model**	**Sliding Window Option**	**Parameters**	**Test-Accuracy**
1	Keras Sequential Neural Network	3	Hidden layers: 2 (21, 21 neurons)	0.5625
2	Multi-layer Perceptron	1	Hidden layers: 2 (52, 32 neurons)	0.5345
3	Decision Tree	1	Depth = 4	0.5295

Table 6.1.: Comparison of classifiers

As you can see in the table Table 6.1, too, you can not even tell, that the more features you use, the better the outcome will be. For example with the Multi-layer Perceptron you get a worse accuracy with more features. But for the actual situation we recommend

to use the first ranked model for predicting the outcome for football games, but we will maybe use a different model with other features in the future.

6.2. Predicting the final goals

6.2.1. Quality Criteria

–> TODO Khaled

6.2.2. Results evaluation of machine learning regressors using scikit library

Multioutput regressor(Gradient Bossting Regressor)
The final training accuracy Final is 20.59%, training accuracy home team goals is 17.10% and training accuracy away team goals: 24.08%
The final test accuracy for gradient booster is 18.89% , test accuracy for home team goals is 12.93% and test accuracy for away team goals is 24.86%. Also, the coefficient of determination R square value of the prediction is calculated to be 0.1612.
Multioutput regressor(Support Vector Regressor)
The final training accuracy Final is 16.22%, training accuracy home team goals is 13% and training accuracy away team goals: 19.43%
The final test accuracy for gradient booster is 16.26% , test accuracy for home team goals is 10.51% and test accuracy for away team goals is 22.02%. Also, the coefficient of determination R square value of the prediction is calculated to be 0.0882
Decision Tree regressor
The final training accuracy Final is 23.18%, training accuracy home team goals is 32.53% and training accuracy away team goals: 13.83%
The final test accuracy for gradient booster is 22.37% , test accuracy for home team goals is 30.82% and test accuracy for away team goals is 13.83%. Also, the coefficient of determination R square of the prediction is calculated to be 0.0898
Random Forest Regressor
The final training accuracy Final is 45.47%, training accuracy home team goals is 43.53% and training accuracy away team goals: 47.42%
The final test accuracy for gradient booster is 16.12% , test accuracy for home team goals is 14.06% and test accuracy for away team goals is 18.18%. Also, the coefficient of determination R square of the prediction is calculated to be 0.0845
Multi-layer perceptron regressor(using Adam Optimizer)

The final training accuracy Final is 18.53%, training accuracy home team goals is 14.02% and training accuracy away team goals: 23.04%
The final test accuracy for gradient booster is 18.82% , test accuracy for home team goals is 11.36% and test accuracy for away team goals is 26.28%. Also, the coefficient of determination R square of the prediction is calculated to be 0.1538
Multi-layer perceptron regressor(using Stochastic Gradient optimizer)
The final training accuracy Final is 15.01%, training accuracy home team goals is 0.14% and training accuracy away team goals: 29.88%
The final test accuracy for gradient booster is 15.20% , test accuracy for home team goals is 0.14% and test accuracy for away team goals is 30.26%. Also, the coefficient of determination R square of the prediction is calculated to be negative 0.0015
The r square value of the model must be in between 0 and 1 and as close to 1 is the better. As observed, none of the regressor model using machine learning algorithms has provided us the satisfactory accuracy to use it in backend of our website. Hence, we have decided to use deep learning with tensorflow and keras to perform regression.

6.2.3. Comparing the regressors

This section highlights the different regressor models created for two different datasets with different features and then these models are compared to find the best model to make predictions on the new data or test data, we have created three different models using tried trial and error to set parameters and hiddenlayers units.
In classification tasks, it is easy to calculate sensitivity or specificity of classifier because output is always binary correct classification, incorrect classification. So we can count good/bad answers and based on the confusion matrix calculate some measurements. But in regression tasks, the output is a number. So we can't just say is it correct or incorrect but instead we should measure "how far from true solution we are" by either calculating coefiicient of determination Rsquare or by focusing on minimizing the mean squared error also known as loss. That is why in our regression models, we have calculated loss for validation as well as for training. For all the three models, input units have been taken equal to features that is 21 units and as we are predicting two outputs- Home Team Goals and Away Team Goals, the output layers are having 2 units or neurons. For the dataset sliding02, we have used 3 hidden layers from which first hidden layer has 30 units which makes the neural network densely connected with 20 units in the second hidden layer and 10 in the third hidden layer. While training the model, we have observed the validation accuracy of 68 percent and Training accuracy of 70 percent. For second model, we have

Dataset Sliding02 Models	QM-HG-Train	QM-AG-Train	FinalQM- Train	QM-HG-Test	QM-AG-Test	FinalQM- Test
Model1	0.83	0.84	0.83	0.81	0.83	0.82
Model2	0.82	0.84	0.83	0.81	0.84	0.83
Model3	0.82	0.84	0.83	0.81	0.84	0.83

Table 6.2.: Quality Model for models with different hidden units for dataset sliding02

Dataset Sliding03 Models	QM-HG-Train	QM-AG-Train	FinalQM- Train	QM-HG-Test	QM-AG-Test	FinalQM- Test
Model1	0.83	0.84	0.84	0.81	0.83	0.82
Model2	0.80	0.83	0.82	0.80	0.84	0.82
Model3	0.83	0.84	0.83	0.81	0.84	0.82

Table 6.3.: Quality Model for models with different hidden units for dataset sliding03

used three hidden layers with 21 units in first hidden layer making the artificial neural network as fullyconnected, 10 units in second hidden layer and 5 in third hidden layer which give us validation accuracy of 66 percent and training accuracy of 72 percent. And the third model have 4 hidden layers with 21 units in first hidden layer layer, 14 units in second hidden layer, 12 units in third hidden layer and 10 units in last hidden layer which give us validation accuracy of 56 percent and training accuracy of 69 percent.
Similarly, similar experiments have been made with dataset sliding03 as well. For all the models we have also evaluated the quality for each team and for both training dataset and test dataset using the quality criteria defined in the above section. The evaluated qualities for all three models for dataset sliding02 and sliding03 have displayed in the respective tables Table 6.2 and Table 6.3 where QM is abbreviation of Quality Model, HG is abbreviation of Home goals and AG is abbreviation of Away goals .

We have used first model from dataset sliding02 as our best model as the validation and training accuracy of this model is better than the other models with a percentage of 68 percent and 72 percent respectively,ideally the validation accuracy must be slightly better than the training accuracy to prevent overfitting of the model. Because if the training loss is higher than validation loss then there are chances of overfitting and if the training loss is very less than validation loss then there are chances of underfitting.But still the validation accuracy is quite lower than the training accuracy.
Also, as our approach is to minimize the mse that is mean squared error in order to get predictions on test data as close to actual data, this model provides us with minimum training mse of 0.19 and validation mse of 0.21. Hence, we are considering it as our best model.
Using quality criteria defined, we have observed a final quality of performance for model 1 for Training data is 83 percent and for test data it is 82 percent.One important point is that the quality evaluation for all the models are showing almost similar values to each

other which is not an ideal solution, so there are lot of scope of impovement in the project.

6.2.4. Comparing Regression with Multi Class Classification

As it is described before the Multi Class Classification has still many possibilities to improve, so in this point we only compare the actual model with the best regression model. For this we will use our own Quality Criteria, but as it is described in the part 'Quality Criteria', this has some disadvantages and has to be improved in the future. Because of this, the test accuracy is also used for the comparison. For the Multi Class Classification approach we are reaching a test accuracy of 30.54% for the away team and 37.64% for the home team, which makes an average of 34.09%. With the regression model we are reaching a test accuracy of 69.5%. Using our quality criteria we are reaching and average of 80.38% (77.30% for the home team and 83.47% for the away team) with the Multi Class Classification and 82.5% (83% for the home team and 84% for the away team) with the regression approach. Because of the better results with the regression model the team decided to use this model for the homepage. After improving the regression and the Multi Class Classification this could change.

7. Deployment

7.1. Backend

–> TODO Till

7.2. API

–> TODO Till

7.3. Frontend

–> TODO Khaled

8. Conclusion

We reached all main goals for the project. The best model has a decent accuracy and we have learned many things about machine learning. Through the team work with SCRUM we were able to delegate the tasks in a way, that we get the best end solution. Additionally we learned how to apply SCRUM for team work, which will help us in our further working life. All steps were necessary for the final outcome, this means we did not got stuck in a wrong direction. The project is proper documented, that a future team is able to continue with our work on the project. The biggest issues we had especially in the beginning of the project, because we did not really know how we should reach our goals or what goals we really had. For the future it would be awesome if the communication between the product owners and the students would be better. Not only in how we should start and what are the main goals, but also how the grade will be exactly built and how we have to structure the report. It would be a good way to create a one- or two-paper with all the common guidelines that the students know exactly what the product owners are require and how they will evaluate the outcome. We are not in a stage that we could actually really earn very much money with our model, which means, that we are only hardly over a 50% accuracy. We are still have the goal to improve our model to reach higher accuracies. For the next semester the main part will be improving the model through additional features and through changing the model.

Bibliography

[1] Andrew Carter. *Deep Neural Network (DNN) Football/Soccer Predictor*. URL: https://github.com/AndrewCarterUK/football-predictor.

[2] Andrew Carter; Sergej Dechant. *Sliding Window 01*. 2019. URL: https://github.com/thu-soccer/project/blob/master/code/process_csv_data.py.

[3] Andrew Carter; Sergej Dechant. *Sliding Window 02*. 2019. URL: https://github.com/thu-soccer/project/blob/master/code/process_csv_data_shots.py.

[4] Andrew Carter; Sergej Dechant. *Sliding Window 03*. 2019. URL: https://github.com/thu-soccer/project/blob/master/code/process_csv_data_shots_possession.py.

[5] Sergej Dechant. *Google Colab Notebook for Data Preparation and Tensorflow Neural Networks*. 2019. URL: https://github.com/thu-soccer/project/blob/master/colab/colab_nn.ipynb.

[6] Kaggle Inc. *European Soccer Database*. 2019. URL: https://www.kaggle.com/hugomathien/soccer.

[7] Kenneth Jensen. *IBM SPSS Modeler CRISP-DM Guide*. 2012. URL: ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerCRISPDM.pdf.

[8] Sergej Dechant; Martin Schmid. *Jupyter Notebook Sliding Windows*. 2019. URL: https://github.com/thu-soccer/project/blob/master/code/sliding_window.ipynb.

A. Appendix

A.1. Architectures for various neural nets

Table A.1.: Test Variation of Hidden-Layers and Neurons for Neural Nets

Name	**Input**	**HLayer1**	**HLayer2**	**HLayer3**	**HLayer4**	**Output**
model01_H1_H	13	13	-	-	-	3
model02_H1_H	21	21	-	-	-	3
model03_H1_H	29	29	-	-	-	3
model04_H1_H	25	25	-	-	-	3
model05_H1_H	33	33	-	-	-	3
model01_H1_M	13	9	-	-	-	3
model02_H1_M	21	12	-	-	-	3
model03_H1_M	29	14	-	-	-	3
model04_H1_M	25	13	-	-	-	3
model05_H1_M	33	16	-	-	-	3
model01_H1_L	13	4	-	-	-	3
model02_H1_L	21	5	-	-	-	3
model03_H1_L	29	7	-	-	-	3
model04_H1_L	25	6	-	-	-	3
model05_H1_L	33	8	-	-	-	3
model01_H2_H	13	13	13	-	-	3
model02_H2_H	21	21	21	-	-	3

continued on next page

Name	Input	HLayer1	HLayer2	HLayer3	HLayer4	Output
model03_H2_H	29	29	29	-	-	3
model04_H2_H	25	25	25	-	-	3
model05_H2_H	33	33	33	-	-	3
model01_H2_M	13	9	9	-	-	3
model02_H2_M	21	12	12	-	-	3
model03_H2_M	29	14	14	-	-	3
model04_H2_M	25	13	13	-	-	3
model05_H2_M	33	16	16	-	-	3
model01_H2_L	13	4	4	-	-	3
model02_H2_L	21	5	5	-	-	3
model03_H2_L	29	7	7	-	-	3
model04_H2_L	25	6	6	-	-	3
model05_H2_L	33	8	8	-	-	3
model01_H3_H	13	13	13	13	-	3
model02_H3_H	21	21	21	21	-	3
model03_H3_H	29	29	29	29	-	3
model04_H3_H	25	25	25	25	-	3
model05_H3_H	33	33	33	33	-	3
model01_H3_M	13	9	9	9	-	3
model02_H3_M	21	12	12	12	-	3
model03_H3_M	29	14	14	14	-	3
model04_H3_M	25	13	13	13	-	3
model05_H3_M	33	16	16	16	-	3

continued on next page

Name	Input	HLayer1	HLayer2	HLayer3	HLayer4	Output
model01_H3_L	13	4	4	4	-	3
model02_H3_L	21	5	5	5	-	3
model03_H3_L	29	7	7	7	-	3
model04_H3_L	25	6	6	6	-	3
model05_H3_L	33	8	8	8	-	3
model01_H3_F	13	13	10	7	5	3
model02_H3_F	21	18	13	9	5	3
model03_H3_F	29	22	16	11	6	3
model04_H3_F	25	20	15	11	6	3
model05_H3_F	33	25	19	12	6	3
model01_H4_H	13	13	13	13	13	3
model02_H4_H	21	21	21	21	21	3
model03_H4_H	29	29	29	29	29	3
model04_H4_H	25	25	25	25	25	3
model05_H4_H	33	33	33	33	33	3
model01_H4_M	13	9	9	9	9	3
model02_H4_M	21	12	12	12	12	3
model03_H4_M	29	14	14	14	14	3
model04_H4_M	25	13	13	13	13	3
model05_H4_M	33	16	16	16	16	3
model01_H4_L	13	4	4	4	4	3
model02_H4_L	21	5	5	5	5	3
model03_H4_L	29	7	7	7	7	3

continued on next page

Name	**Input**	**HLayer1**	**HLayer2**	**HLayer3**	**HLayer4**	**Output**
model04_H4_L	25	6	6	6	6	3
model05_H4_L	33	8	8	8	8	3
model01_H4_F	13	13	10	7	5	3
model02_H4_F	21	18	13	9	5	3
model03_H4_F	29	22	16	11	6	3
model04_H4_F	25	20	15	11	6	3
model05_H4_F	33	25	19	12	6	3

end of table

A.2. Daily Scrum Logs

Hochschule Ulm

Titel	Daily Scrum
Date	24.10.2019
Sprint	01
Participants	Sergej, Martin, Lisa, Khaled

1. What have you done?

   - Lisa: Installed TF, Articles about TF, Exercises in TF, Data Profiling, Python ML
   - Khaled: Keras & TF Anaconda Tutorials, Book about NN
   - Sergej: Nielssens Book NN, Installed TF & Keras, Tutorial for TF, Look at Soccer DB SQLite, Reading on Data Science, Research on Data Science Maths, Basic research on Linear Regression
   - Martin: Keras & TF + Installation

2. What will you do?

   - Lisa: Installing Keras, Do Examples, Research about ML
   - Khaled: Neuroal Network, Data Profiling
   - Sergej: Get used to TF for SQL based data
   - Martin: Data Profiling, getting used to TF/ Keras

3. Issues?

   - Lisa: -
   - Khaled: -
   - Sergej: -
   - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	31.10.2019
Sprint	1
Participants	Khaled, Lisa, Sergej, MArtin

1. What have you done?
    - Lisa: Read Articles about keras + videos, installation, Example for Keras and TF
    - Khaled: data profiling, started data pre-processing (missing data), started using sci-kit learn library, Research Neuronal Network
    - Sergej: reading on classification, linear regression, handwriting recognition, data profiling, made some examples with TF
    - Martin: research for keras & tf, data profiling
2. What will you do?
    - Lisa: Do more examples, do some tests with our database
    - Khaled: continue data processing
    - Sergej: data profiling, research
    - Martin: data profiling & preprocessing
3. Issues?
    - Lisa: -
    - Khaled: -
    - Sergej: -
    - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	07.11.2019
Sprint	Sprint 02
Participants	Khaled, Sergej, Martin

1. What have you done?

   - Lisa: Analyzed the Dataset
   - Khaled: Started analysing the data
   - Sergej: Spent time on data profiling, literature research on dp, slice & dice (pivot tables)
   - Martin: Research on feature selection, data profiling

2. What will you do?

   - Lisa: feature selection
   - Khaled: Starting to use both teams as input for analytics
   - Sergej: write some pages for the report on data profiling
   - Martin: research on feature selection, data profiling

3. Issues?

   - Lisa:-
   - Khaled: Maybe using google collab?
   - Sergej: -
   - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	14.11.2019
Sprint	Sprint 02
Participants	Khaled, Sergej, Martin, Lisa

1. What have you done?

   - Lisa: research on feature selection, found tutorial and diagramm
   - Khaled: continued data pre-processing, mapping feature for season
   - Sergej: data profiling + documentation
   - Martin: extracted reduced dataset from Bundesliga seasons 2014/15 and 2015/16, SQL, research

2. What will you do?

   - Lisa: do the tutorial, learn more about fs
   - Khaled: find good features for hidden layers
   - Sergej: data profiling + documentation
   - Martin: research on feature selection, normalization, prediction, NN

3. Issues?

   - Lisa: database is very large
   - Khaled: database is very large
   - Sergej: -
   - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	21.11.2019
Sprint	Sprint 02
Participants	Khaled, Sergej, Martin, Lisa

1. What have you done?

   - Lisa: did the tutorial, analyzed xml-columns, research feature selection
   - Khaled: handled high correlation features, handled NaN/null-values, provided model
   - Sergej: started report LaTeX, research and trials of feature selection
   - Martin: documentation of data extraction, research feature selection and normalization

2. What will you do?

   - Lisa: evaluate the model, find some new features, apply possession-feature
   - Khaled: include possession feature
   - Sergej: maybe try different model
   - Martin: normalization, feature selection

3. Issues?

   - Lisa: -
   - Khaled: -
   - Sergej: waiting for the VM
   - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	28.11.2019
Sprint	03
Participants	Khaled, Lisa, Sergej, Martin

1. What have you done?

   - Lisa: implemented time window, extracted average amount of points feature
   - Khaled: average goal feature, online course for model implementation, looked on possession feature
   - Sergej: reading on feature extraction and how to adjust a model in the right way
   - Martin: extract feature avg. earned points, Reading raschka ebook python ml

2. What will you do?

   - Lisa: trying to make a model, find extract the features differently
   - Khaled: continue research
   - Sergej: implement additional sliding window approach
   - Martin: Research, continue the Raschka book

3. Issues?

   - Lisa: -
   - Khaled: -
   - Sergej: -
   - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	05.12.2019
Sprint	03
Participants	Khaled, Lisa, Sergej, Martin

1. What have you done?
    - Lisa: tried to build a TF Model, read Raschka
    - Khaled: finished certificate about deep learning with TF by IBM,
    - Sergej: implemented sliding window, tried to build a model with TF,
    - Martin: Normalization/Standardization Raschka

2. What will you do?
    - Lisa: Look at sliding window from Sergej, feature extraction
    - Khaled: try logostic regression and learned from certificate
    - Sergej: TF Serving, use sliding window, try different classifiers
    - Martin: extract the XML-feature shot-on-goal

3. Issues?
    - Lisa: -
    - Khaled: -
    - Sergej: -
    - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	12.12.2019
Sprint	03
Participants	Khaled, Lisa, Sergej, Martin

1. What have you done?

   - Lisa: built model with scikit learn, accuracy of 52%, understanding of sergejs sliding window, setup old LaTeX template
   - Khaled: worked on logistic regression with TF
   - Sergej: spent more time on sliding window, played around with TF
   - Martin: feature extraction from xml columns

2. What will you do?

   - Lisa: start with the report and more feature selecting, try to build a model
   - Khaled: analyzing the data
   - Sergej: get understanding of how to use the model
   - Martin: feature selection

3. Issues?

   - Lisa: -
   - Khaled: -
   - Sergej: -
   - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	19.12.2019
Sprint	04
Participants	Khaled, Lisa, Sergej, Martin

1. What have you done?

- Lisa: made a NN with Sklearn with 53% acc, tried model with TF, continued the Report
- Khaled: Did some Decision Trees, worked on architecture
- Sergej: fixed an error in the data for chronological order, added feature to martins code, tried KNN found optimal K with 51%
- Martin: started extraction of ball possession feature

2. What will you do?

- Lisa: take historical ordered dataset and try Sklearn and TF model again with new data, continue report
- Khaled: keep working on the classifiers and on the architecture
- Sergej: design a couple of NN, use Schwerins list of most common used NN's
- Martin: extract ball possession in minutes per match, try Schwerins list of common used NN's

Hochschule Ulm

3. Issues?
    - Lisa: -
    - Khaled: -
    - Sergej: -
    - Martin: -

Hochschule Ulm

Titel	Daily Scrum
Date	09.01.2020
Sprint	04
Participants	Khaled, Lisa, Sergej, Martin

1. What have you done?

   - Lisa: Finished reading raschka ML book, continued with the report, tried the sklearn model and decision tree with historical ordered data,
   - Khaled: Finished DecissionTrees for 2 new options, research of multy class classifications NN, coding review
   - Sergej: wrote some code, built some models, researched and implemented keras + TF + google colab
   - Martin: comparison of different amount of hidden layers and number of neurons per layer, extract ball possession in minutes per match

2. What will you do?

   - Lisa: finishing the report and create the presentation
   - Khaled: finishing the report and create the presentation
   - Sergej: finishing the report and create the presentation
   - Martin: finishing the report and create the presentation

3. Issues?

   - Lisa: -
   - Khaled: -
   - Sergej: -
   - Martin: -

A.3. Report of First Semester

Hochschule Ulm

Masterproject

Geocoding and Routing with Pelias and Valhalla

Contributors:
Martin Schmid, Sergej Dechant

Expert: Prof. Dr. von Schwerin

Expert: Prof. Dr. Herbort

Expert: Prof. Dr. Goldstein

Thursday 19th September, 2019

Contents

Chapter 1

Project Presentation and Scope

1 Introduction

The purpose of this paper is to document the progress of the "junior team" during the first half of the data science project in form of a technical report. Moreover, this report should allow readers to gain an understanding of the topics covered in the data science project as well as be able to reproduce and extend the developed and utilized solutions. The covered tasks during the first half of the project can be categorized into three main areas:

1. Infrastructure
    - Set up a virtual machine (Ubuntu Linux)
    - Install and configure Pelias and Elasticsearch
    - Install and evaluate different routing engines
2. Data acquisition and preparation
    - Gather postcode data of European countries from different sources
    - Merge postcode data into a single data source for Pelias and Elasticsearch
3. Geocodoing and Routing
    - Test geocoding with Pelias based on precalculated two-digit postcode centroids
    - Test routing between two-digit postcode centroids wit a routing engine

2 Requirements

The main requirements were to evaluate Pelias as an open source geocoding service and as an alternative to Nominatim as well as to realize routing from one two-digit postcode to another. In order to achieve this it was necessary to build a database of postcodes and create a map of Europe based on data provided by Openstreetmaps, Whosonfirst, Geonames and Postcode-info. Furthermore, routing engines as an alternative to Graphhopper had to be evaluated. Last but not least an adequate documentation on how these requirements can be fulfilled and the outcome reproduced had to be written.

Chapter 2

Pelias

1 General

1.1 Capabilities

Pelias is a software solution/library used for geocoding. Geocoding is the process of taking input text, such as an address or the name of a place and returning a latitude/longitude location on the Earth's surface for that place. The "senior team" used Nominatim for geocoding, which is a tool for geocoding just like pelias. One of our main tasks in the course of the first half of the project was to test and evaluate pelias as an alternative open-source geocoder to Nominatim.

Here are some benefits of the pelias API [8]:

- Completely open-source and MIT licensed
- A powerful data import architecture: Pelias supports many open-data projects out of the box but also works great with private data
- Support for searching and displaying results in many languages
- Fast and accurate autocomplete for user-facing geocoding
- Support for many result types: addresses, venues, cities, countries, and more
- Easy installation with minimal external dependencies

As mentioned above, pelias has the ability to import data from many different open-data projects as well as own private data. The importers filter, normalize, and ingest geographic datasets into the Pelias database. Currently there are five officially supported importers [8]:

- **OpenStreetMap:** supports importing nodes and ways from OpenStreetMap
- **OpenAddresses:** supports importing the hundreds of millions of global addresses collected from various authoritative government sources by OpenAddresses
- **Who's on First:** supports importing admin areas and venues from Who's on First
- **Geonames:** supports importing admin records and venues from Geonames
- **Polylines:** supports any data in the Google Polyline format. It's mainly used to import roads from OpenStreetMap
- **Custom Data Importer:** creates a Pelias record for each row in a CSV file. Each row must define a source, latitude, longitude, and either an address, name, or both. This feature was used to import two-digit postcodes into Pelias which will be described in chapter Data Acquisition and Preparation.

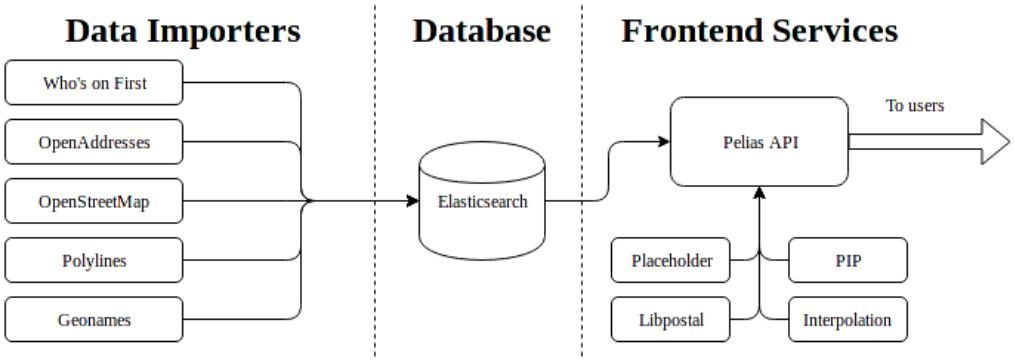

Figure 1.1: Overview of the pelias architecture

1.2 Database

The underlying datastore that powers the search results and does query-lifting is Elasticsearch. Currently version 2.4 and version 5 is supported, with plans to support Elasticsearch 6 soon. The developers built a tool called pelias-schema that sets up Elasticsearch indices properly for Pelias.

2 System Requirements

2.1 Software Requirements

- **Node.js:** Version 8 or newer is required, version 10 is recommended for improved performance.
- **Elasticsearch:** Version 2.4 or 5.6
- **SQLite:** Version 3.11 or newer
- **Libpostal:** Pelias relies heavily on the Libpostal address parser. Libpostal requires about 4GB of disk space to download all the required data.

2.2 Hardware Requirements

- At a minimum 50GB disk space to download, extract, and process data
- 8GB RAM for a local build, 16GB+ for a full planet build. Pelias needs a little RAM for Elasticsearch, but much more for storing administrative data during import
- As many CPUs as possible. There's no minimum, but Pelias builds are highly parallelizable, so more CPUs will help make it faster.

Actual system used for the project (Europe build):

- 1 virtual machine (Ubuntu Linux) with 64 GB RAM, 500GB HDD, 4 CPU cores

- RAM utilization is at 30 GB, however during the import of openstreetmaps data and calculating polylines from it up to 40 GB of RAM were used. Imports and calculations maxed out all CPU cores. It is possible to reduce the required amount of RAM for imports and calculations. However, this requires splitting up openstreetmap files in smaller files with other tools beforehand.
- Including "raw data" (before the import and calculations) around 400GB of data are persisted on HDD, Elasticsearch uses 100GB.

3 Installation and Configuration

Pelias can be installed with Docker Images, manually from scratch or with Kubernetes. For testing purposes installing Pelias using Docker Images is strongly recommended by the developers [7]. Pelias can also be installed manually from scratch, but due to the large amount of dependencies this is not recommended by the developers. To use Pelias in production, the development team suggests an installation with Kubernetes, which is by far the most tested and best way to install and use Pelias in production according to the development team.

3.1 Installation with Docker

On the virtual machine Pelias was installed and maintained with Docker and Docker-Compose. Install Docker and Docker-Compose:

```
sudo apt-get update
sudo apt-get install \
apt-transport-https \
   ca-certificates \
        curl \
        gnupg-agent \
        software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
    | sudo apt-key add -
sudo add-apt-repository \ "deb [arch=amd64] https://
   download.docker.com/linux/ubuntu \
        $(lsb_release -cs) \ stable"
```

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd
    .io
sudo groupadd docker
sudo usermod -aG docker $USER
sudo systemctl enable docker
sudo curl -L "https://github.com/docker/compose/
    releases/download/1.24.0/docker-compose-$(uname -s)-
    $(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Afterwards Pelias can be installed by cloning Pelias' git repository. In this repository Pelias' developers provide example projects (e.g. Beligum, Portland Metro, etc.). Pelias' "planet" project was used as a starting point for a Europe build. For this Pelias was forked on Github and cloned onto the VM. The project can be found in the following folder:

```
/home/dataproject/git/pelias-docker/projects/Europe
```

In order to build and run Pelias with data for Europe four configuration files in this folder are needed:

1. .env
2. Elasticsearch.yml
3. pelias.json
4. docker-compose.yml

The files can be found in the appendix on page 29.

In .env DATA_DIR and DOCKER_USER are important entries/variables. DATA_DIR specifies where Pelias will store downloaded data and build its other services. DOCKER_USER specifies the user id. This user id will be used for accessing files on the host filesystem in DATA_DIR since Pelias' processes run as non-root users in containers. In Elasticsearch.yml both thread pool sizes had to be increased since the default values were too small. Pelias importers delivered too much data concurrently for Elasticsearch which resulted in corrupted data. In pelias.json all Pelias services are configured. These services run as docker containers. Therefore, it is not necessary to provide complete full paths on the host filesystem or IP/DNS addresses.

Paths are mapped to the paths provided in the docker compose file and .env file. Docker has its own networking and DNS. Services in a docker network can be addressed by using docker compose service names as well as container names and ids. Container ports can be mapped to host ports. The variables DOCKER_USER and DATA_DIR in docker-compose.yml are mapped to the corresponding entries in .env. Inside containers pelias.json is made available in /code/pelias.json. Ports are mapped in the following way: hostport:containerport. The "image" directive tells docker from where it has to pull the container image. In this case all images are pulled from the Pelias repository on Docker-Hub. After the colon a tag is specified (e.g. master or a version/hash). If no tag is provided, the latest version will be pulled. With this configuration it is possible to build Europe completely with the following commands and order (cd to Europe project folder first):

```
pelias compose pull
pelias elastic start
pelias elastic wait
pelias elastic create
pelias download all
pelias prepare all
pelias import all
pelias compose up
```

3.2 Installation from scratch

In order to do a clean installation of the pelias service and its dependencies on a production server at a later point in time we decided to try the installation from scratch and wrote an installation guide. The complete guide can be found in the appendix on page 38. We did the installation on a Linux VM running Ubuntu 18.04.

Installing Dependencies

Node.js: Version 8 or newer required, version 10 recommended

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -
    E bash -
sudo apt-get install -y nodejs
```

Elasticsearch: Version 2.4 or 5.6

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-
    elasticsearch | sudo apt-key add -
echo "deb https://artifacts.elastic.co/packages/5.x/apt
    stable main" | sudo tee -a /etc/apt/sources.list.d/
    elastic-5.x.list
sudo apt update && sudo apt upgrade
sudo apt install apt-transport-https uuid-runtime pwgen
     openjdk-8-jre-headless
sudo apt-get update
sudo apt update
sudo apt install elasticsearch
```

SQLite: Version 3.11 or newer

```
sudo apt-get update
sudo apt-get install sqlite3
sqlite3 --version
sudo apt-get install sqlitebrowser
```

Libpostal: In order to install libpostal you will have to manually compile the source code.

```
sudo apt-get install curl autoconf automake libtool pkg
    -config
cd /
git clone https://github.com/openvenues/libpostal
cd libpostal
./bootstrap.sh
./configure --datadir=[...some dir with a few GB of
    space...]
make -j4
sudo make install
sudo ldconfig
```

Installing Pelias

Once you are done installing all the dependencies and downloaded the data for your pelias build you can start installing pelias itself.

```
for repository in schema whosonfirst geonames
    openaddresses openstreetmap polylines api
```

```
    placeholder interpolation pip-service; do
git clone https://github.com/pelias/${repository}.git #
    clone from Github
pushd $repository > /dev/null # switch into importer
    directory
npm install # install npm dependencies
popd > /dev/null # return to code directory
done
```

After the installation you will have to set up the elasticsearch schema in order to use pelias.

```
cd /pelias/schema # assuming you have just run the bash
    snippet to download the repos from earlier
./bin/create_index
```

Chapter 3

Data Acquisition and Preparation

1 Two-Digit Postcodes

CSV Data provided by geonames.org [9] and postcode.info, which was scrapped and provided by a fellow student [5], was used as basis for calculating two-digit postcodes for European countries. At the first iteration two-digit postcodes were calculated from Geonames data. In order to achieve this Python scripts were written [1]. These scripts process a Geonames CSV file and provide a new CSV file with two-digits postcodes including their centroids. Here is an example of six calculated two-digit postcodes in Germany:

DE80	geonames2d	80	postalcode	48.1615	11.5509
DE81	geonames2d	81	postalcode	48.1254	11.5726
DE82	geonames2d	82	postalcode	47.911	11.2502
DE83	geonames2d	83	postalcode	47.8713	12.2803
DE84	geonames2d	84	postalcode	48.4086	12.4327
DE85	geonames2d	85	postalcode	48.4174	11.6308

Table 3.1: Geonames Two-Digit Postcodes

At the second iteration Geonames and Postcode data were combined after having done some preparation steps on the Postcode data. Again Python scripts were written [2]. Here is an excerpt:

DE80	geonamesandpostcodeinfo	80	postalcode	48.1512	11.5938
DE81	geonamesandpostcodeinfo	81	postalcode	48.1331	11.6046
DE82	geonamesandpostcodeinfo	82	postalcode	47.9419	11.2759
DE83	geonamesandpostcodeinfo	83	postalcode	47.888	12.2627
DE84	geonamesandpostcodeinfo	84	postalcode	48.4367	12.4206
DE85	geonamesandpostcodeinfo	85	postalcode	48.4171	11.6294

Table 3.2: Geonames and Postcode.info Two-Digit Postcodes

Comparing these two tables one can see that the coordinates changed slightly. This is due to the fact that now two different data sources were used for the calculation of centroids resulting in more accurate coordinates. Postcodes in Malta were aggregated and their centroids calculated using the first three digits as requested by one of the project's experts. Finally the calculated two-digit postcodes had to be imported into Pelias. For this a Python script, which creates a CSV file conforming to Pelias' custom data importer, had to be written [2]. This CSV file has to be copied to the following path, which is defined in docker-compose.yml and .env: /data/pelias-docker-compose/geonamesandpostcodeinfo/geonamesandpostcodeinfo2Dpostalcodes.csv Afterwards the command "pelias import csv" has to be run. Once the import is complete, the custom layer "geonamesandpostcodeinfo" and its data can be queried as follows:

`http://141.59.29.110:4000/v1/search?text=DE81&sources=geonamesandpostcodeinfo`

This query delivers the following JSON file:

```
"type": "FeatureCollection",
   "features": [
                {
       "type": "Feature",
       "geometry": {
          "type": "Point",
          "coordinates": [
               11.6046,
               48.1331
       ]
```

```
        },
"properties": {
        "id": "1141",
        "gid": "geonamesandpostcodeinfo:postalcode:1141",
        "layer": "postalcode",
        "source": "geonamesandpostcodeinfo",
        "source_id": "1141",
        "name": "DE81",
        "confidence": 1,
        "match_type": "exact",
        "distance": 690.624,
        "accuracy": "centroid",
        "country": "Germany",
        "country_gid": "whosonfirst:country:85633111",
        "country_a": "DEU",
        "region": "Bayern",
        "region_gid": "whosonfirst:region:85682571",
        "region_a": "BY",
        "macrocounty": "Oberbayern",
        "macrocounty_gid": "whosonfirst:macrocounty:404227567",
        "county": "Muenchen",
        "county_gid": "whosonfirst:county:102063261",
        "county_a": "MN",
        "locality": "Muenchen",
        "locality_gid": "whosonfirst:locality:101748479",
        "neighbourhood": "Haidhausen",
        "neighbourhood_gid": "whosonfirst:neighbourhood:85905613",
        "continent": "Europe",
        "continent_gid": "whosonfirst:continent:102191581",
        "label": "DE81, Muenchen, Germany"
                }
```

```
        }
],
"bbox": [
        11.6046,
        48.1331,
        11.6046,
        48.1331
        ]
}
```

Chapter 4

Routing Engines

1 General

The Pelias API and Pelias services are only suited for the purpose of geocoding and reverse geocoding. Geocoding retrieves coordinates (latitude and longitude) for a given address or postode and reverse geocoding finds the nearest known address or postcode for a provided pair of latitude and longitude. In order to find the shortest or fastest route between two given addresses Pelias has to be used in conjunction with a routing engine. The two addresses are fed into Pelias and Pelias provides coordinates for them which are then used as input values for finding a route from one coordinate to the other using a routing engine and the metrics inside the routing engine. The senior team used the routing engine Graphhopper in connection with their geocoding service Nominatim. Graphhopper as you will see later in this report is a very good and fast routing engine, however the developers of the Pelias service recommend using the routing engine Valhalla which is developed by the same company (Mapzen) as Pelias and therefore has better service interoperability with Pelias than any other routing engine.

1.1 Comparison of Routing Engines

Part of this project was to research and evaluate possible routing engines for Pelias. Internet research conducted by the junior them revealed a comparison of open source routing engines which was done by one of the members of Openstreetmaps. The following two figures illustrate the required computing

time (in ms) to calculate a route depending on the length of the route (in km)[6]:

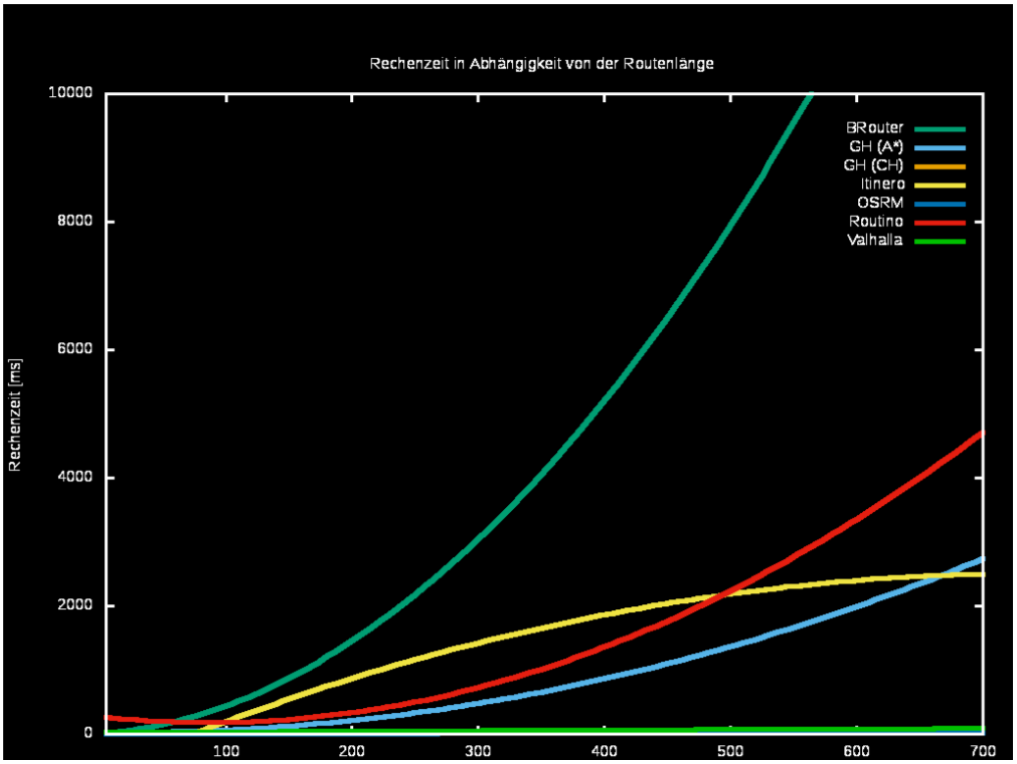

Figure 1.1: Comparison of all open source Routing Engines

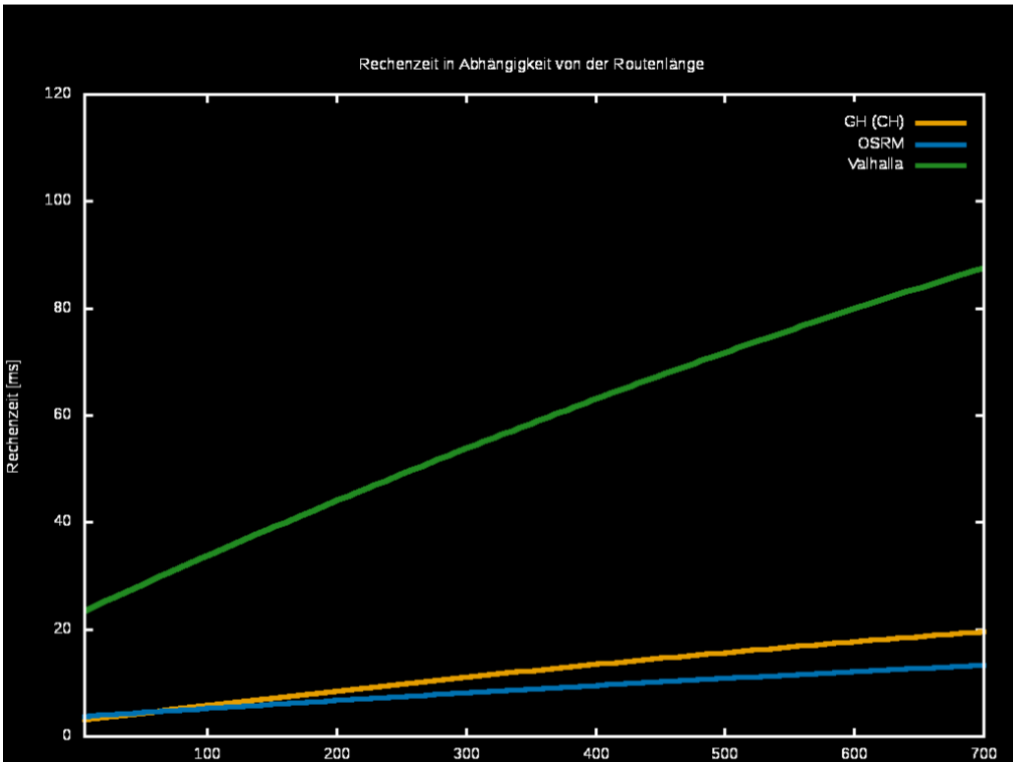

Figure 1.2: Comparison of fastest open source Routing Engines

As can be seen, Graphopper (GH), Open Source Routing Machine (OSRM) and Valhalla are the best-performing open source routing engines. Therefore, a closer look can be taken at those three in the table 4.1.

Table 4.1: Comparison of Routing Engines

Comparison Criteria	**Graphhopper**	**Valhalla**	**OSRM**
License	Apache-License (proprietary in parts)	MIT license	BSD license
OS	Java (also Android, iOS)	C++, Apple/Linux	C++ (NodeJS), Apple/Linux/Windows
Continued on next page			

Table 4.1: Comparison of Routing Engines (Continued)

Comparison Criteria	**Graphhopper**	**Valhalla**	**OSRM**
Algorithm	Contraction Hierarchies, Dijkstra/A*, Hybrid	A* with individual improvements	Contraction Hierarchies
Documentation & Setup	Good documentation, 'quick start'	Good documentation, 'quick start', ubuntu repository with web-frontend	Good documentation, setup with docker or self-compiled
Routing-features	Turn restriction (A*), guidepost, alternatives, height data optional	Turn restriction, guidepost, height data optional	Turn restriction (A*), driving lanes, guidepost, alternatives, no height data
Special Features	Track Matching, cost != time, TSP with jsprit	Tile-based data storage, dynamic cost, matrix, isochrones, intermodal, Designed for working with OpenStreetMap	Matrix, track matching, TSP, data tiles, cost != time

Unfortunately OSRM has very high hardware requirements[3]. Preprocessing the car profile requires at least 175 GB of RAM and 280 GB of disk space. Additionally, 35 GB are needed for the planet osm.pbf (Openstreetmaps) and 40 to 50 GB for the generated data files. For the foot profile 248 GB of RAM are needed. During runtime the car profile requires around 64 GB of RAM, the foot profile even more. Basically OSRM loads the preprocessed files completely into RAM[3]. The project team's VM had only 64 GB of RAM and half of it was already used for Pelias and Elasticsearch. Hence, it was not possible to install OSRM and evaluate it completely.

1.2 Graphopper vs Valhalla

We performed an in-depth comparison of the time it takes for routing from one point to another with Graphhopper and Valhalla. The results of these test series can be seen in the table 4.2.

Table 4.2: Graphhopper vs. Valhalla test row

		Route 1	**Route 2**	**Route 3**	**Route 4**	**Route 5**	**Route 6**
	Route from	Catania, Italy Latitude: 37.502236 Longitude: 15.08738	1100-148 Lisbon, Portugal Latitude: 38.707751 Longitude: -9.136592	Ulm, Baden-Württemberg, Germany Latitude: 48.3974 Longitude: 9.993434	Paris, France Latitude: 48.856697 Longitude: 2.351462	37011 Bardolino VR, Italy Latitude: 45.553553 Longitude: 10.637519	North Cape, E 69, Norway Latitude: 71.169951 Longitude: 25.785889
	Route to	1357 Copenhagen, Denmark Latitude: 55.686724 Longitude: 12.570072	Warsaw, Warszawa, Poland Latitude: 52.231924 Longitude: 21.006727	Munich, Bavaria, Germany Latitude: 48.137108 Longitude: 11.575382	Venice, Venezia, Italy Latitude: 45.437191 Longitude: 12.33459	Gunterstraße 8, 70191 Stuttgart, Germany Latitude: 48.806576 Longitude: 9.178105	89032 Bianco RC, Italy Latitude: 38.087176 Longitude: 16.148511
GH	**first try**	real 0m0.735s user 0m0.013s sys 0m0.006s	real 0m0.300s user 0m0.016s sys 0m0.006s	real 0m0.031s user 0m0.004s sys 0m0.011s	real 0m0.083s user 0m0.016s sys 0m0.000s	-	real 0m0.261s user 0m0.013s sys 0m0.011s
	second try	real 0m0.188s user 0m0.019s sys 0m0.000s	real 0m0.407s user 0m0.014s sys 0m0.005s	real 0m0.022s user 0m0.014s sys 0m0.000s	real 0m0.042s user 0m0.014s sys 0m0.004s	-	real 0m0.216s user 0m0.024s sys 0m0.004s

Continued on next page

Table 4.2: Graphhopper vs. Valhalla test row (Continued)

		Route 1	**Route 2**	**Route 3**	**Route 4**	**Route 5**	**Route 6**
	distance (in km)	2753	3318	139	1113	-	5112
VH	**first try**	real 0m2.098s user 0m0.014s sys 0m0.013s	real 0m4.805s user 0m0.013s sys 0m0.021s	real 0m0.668s user 0m0.012s sys 0m0.006s	real 0m3.121s user 0m0.020s sys 0m0.003s	real 0m1.037s user 0m0.012s sys 0m0.009s	real 0m1.784s user 0m0.030s sys 0m0.003s
	second try	real 0m0.279s user 0m0.018s sys 0m0.008s	real 0m0.498s user 0m0.030s sys 0m0.003s	real 0m0.083s user 0m0.014s sys 0m0.004s	real 0m0.571s user 0m0.014s sys 0m0.008s	real 0m0.086s user 0m0.017s sys 0m0.005s	real 0m0.607s user 0m0.026s sys 0m0.008s
	distance (in km)	2682	3408	141	1116	579	4996
	remark					Coordinates of the starting point in the middle of lake garda. Graphhopper couldn't calculate a route	

The API requests were executed directly on the Graphhopper and Valhalla host machines using the command line programs time and curl. We

can see, that Graphhopper compared to Valhalla does a significantly better job when calculating a new route for the very first time. The execution time in Valhalla varies from the first calculation to the second calculation by up to the factor of ten. This means calculating a route or part of it, which has already been calculated before, the execution time is almost ten times faster compared to the first calculation. Valhalla achieves this with caching routes in RAM. Graphhopper however has a problem, if there is no road or street to be routed from or to for a given start- or end-point. Valhalla in this case just takes the closest routable point instead. Choosing one routing engine over the other depends on the goal which should be achieved. For fastest execution time (not regarding first or second execution) Graphhopper fits best. If you want to make sure, that you receive a route whichever point you calculate from or to, then it is recommended to use Valhalla.

1.3 Conclusion

OSRM is the fastest routing engine on the open-source market. But because of the very high memory requirements of OSRM it is not suitable for the use-case of our project and the cost/benefit-factor is too low. Valhalla would be a good alternative to Graphhopper, because it is compared to other routing engines nearly as fast as Graphhopper and is designed to work with Openstreetmaps-data and also recommended by the Pelias developers to be used in connection with Pelias as a geocoder. Also Valhalla is capable of routing from or to points, which do not have a road or street directly nearby. A very valuable feature especially for two-digit postcode centroids, which Graphhopper does not have. However, in this early state of the project Graphhopper totally fits all the needs and therefore there is no need in replacing Graphhopper with Valhalla.

2 Valhalla

Valhalla was installed and configured according to the official documentation on Github [4]. Tiles and polylines were calculated using the same openstreetmaps pbf file (Europe) which was already used for Pelias. Routing can be achieved by querying Valhalla's api:

```
curl http://141.59.29.110:8002/route --data '{"
    locations":[{"lat":48.1331,"lon":11.6046,"type":"
```

```
break"},{"lat":47.9419,"lon":11.2759,"type":"break
"}],"costing":"auto","directions_options":{"units":"
km"}}' | jq '.'
```

Result:


```
  "summary": {
    "max_lon": 11.605507,
    "max_lat": 48.133167,
    "time": 2397,
    "length": 38.536,
    "min_lat": 47.943157,
    "min_lon": 11.260186
  },
  "locations": [
    {
      "original_index": 0,
      "type": "break",
      "lon": 11.6046,
      "lat": 48.133099,
      "side_of_street": "right"
    },
    {
      "original_index": 1,
      "type": "break",
      "lon": 11.2759,
      "lat": 47.941898,
      "side_of_street": "right"
    }
  ]
```


The complete output and routing instructions are in the appendix page 49.

Chapter 5

Conclusion and Outlook

The requirements described in chapter 1 section 2 were achieved completely. We installed Pelias and all of its' services as a docker image as well as a from scratch installation. The Team built a complete map of Europe based on data provided by OpenStreetmaps, WhosOnFirst and OpenAddresses. Furthermore, we calculated tiles and polylines based on OpenStreetmaps data. This process initially failed, since calculating polylines for complete Europe requires more than 32 GB of RAM-Memory. After an upgrade of the virtual machine to 64 GB this step finished successfully. During the calculation and import steps several bugs in Pelias were found and submitted. Luckily, they were fixed, or workarounds were provided within a few days.
The postcode data was gathered from Geonames.org and Postcode.info. The data from Geonames.org can be downloaded as ZIP-files separated in Countries, or as one single ZIP-file which contains the whole world. The data from Postcode.info had to be scraped from their website by our fellow student Ankur Mehra. After the data from Geonames.org and Postcode.info had been merged, we could calculate the 2-digit postcodes and import our newly generated data basis as a custom data-source into Pelias and Elasticsearch. On this data basis we can now do geocoding in Pelias for 2-digit postcodes in Europe. We did research on several routing engines and decided to use Valhalla as an alternative to the already existing Graphhopper which is used by the senior team. Valhalla offers similar performance for cached data/routes (including start and end points in the vicinity of previously used coordinates) as Graphhopper with a good trade-off between resource requirements and performance/time to calculate a route. Furthermore, polylines calculated from OpenStreetmaps data with Pelias' tools can be used in Valhalla

and vice versa.
All in all, we can say, that the first semester of our project has been successful, and we reached our overall goal of testing Pelias as an alternative to Nominatim. Most of our tasks were finished, although some tasks couldn't be finished during the last sprint. Those tasks are mainly concerned about a VPN connection to the cluster of computers located at campus Albert-Einstein-Allee. As soon as those tasks are finished successfully, we could proceed with installing an instance of our Pelias build on the cluster and run this build as some kind of production-system. However in the coming project phase we will probably be more focused on data analytics and machine learning depending on the product owners requirements.

Bibliography

[1] Sergej Dechant. Geonames 2D Postalcodes, 2019.

[2] Sergej Dechant. Pelias Custom Data Import, 2019.

[3] Daniel J. OSRM Disk and Memory Requirements, 2017.

[4] Greg Knisely. Valhalla, 2019.

[5] Ankur Mehra. Postcode Scraper, 2019.

[6] Frederik Ramm. Routing Engines für OpenStreetMap, 2017.

[7] Julian Simioni. Pelias Documentation, 2018.

[8] Julian Simoni. Pelias API, 2018.

[9] Unxos. Geonames Download, 2019.

B Illustration Directory

List of Tables