Università degli Studi di Padova

Department of Mathematics "Tullio Levi-Civita"

Master Degree in Computer Science

Prediction of football players' position using Data Mining and Machine Learning techniques

Master degree thesis

Supervisor
Prof. Annamaria Guolo
University of Padua

Graduate student
Alberto Gobbo

Academic Year 2022-23

Alberto Gobbo: *Prediction of football players' position using Data Mining and Machine Learning techniques*, Master Degree thesis in Computer Science, Academic Year 2022-23, © February 2023.

“Coloro che fanno sforzi continui sono sempre pieni di speranza.
Abbracciate i vostri sogni e inseguiteli.
Gli eroi quotidiani sono quelli che danno sempre il massimo nella vita.”

Roberto Baggio

Abstract

This thesis focuses on the implementation of Data Mining and Machine Learning techniques for predicting the best position of footballers in the pitch.
The dataset used to this aim has been created starting from the characteristics about professionals football players available from *FIFA22* videogame data.
Since the position held by footballers in the pitch has different levels, classification instruments have been used for data analysis and predictions. Data Mining techniques, including Multinomial Logistic Regression, Discriminant Analysis, and regularization methods, such as Ridge Regression and Lasso, have been adopted for the aim of discovering relationships between the predictors. Machine Learning techniques used mainly for predictions purposes include Decision Tree, Random Forest, k-Nearest Neighbour, Naive Bayes, and Support Vector Machine. In addition to that, the reduction of the response variable classes is considered to check possible improvements on the best Data Mining and Machine Learning techniques found. A comparison between the methods in terms of performance, accuracy of the results and computational cost concludes the analysis.

Contents

List of Figures

List of Tables

Listings

1 Introduction

This chapter explains the general problem that the thesis intends to address, and relates it to the existing literature. Furthermore, all the programming languages and tools used to conduct the thesis work and a brief summary of the thesis structure will be mentioned.

1.1 The problem

Football is one of the most watched sports in the world, where television broadcast share reaches its highest peaks during the Champions League and the World Cup. The business around football is ever increasing as well as the interest of the scientific world in Football Analytics. More in detail, Football Analytics is a discipline which aims to collect and analyze data about matches (see *Football Analytics*). This discipline has grown with the advent of big data and it has given rise to more sophisticated systems for the reading of match statistics, going into detail also in the performances of the players.
For what concerns the football game, two teams face off in a grass pitch where each team is composed by eleven players, one defending its goal (the goalkeeper) and ten outfield players with different roles on the pitch. The target of each team is to score at least one more goal than the opponent team, in order to win the match. Each team is headed by a coach (a.k.a. manager), the main figure responsible for deciding which players to line up and their position on the pitch. Apart from the goalkeeper, the decision of the position of a player is quite tricky. Indeed, the coach tries to place a player according to what he has watched during

the training sessions, trusting on his eyes, or basing his decision on the positions the player has held in his career. Sometimes, subjective feelings of the coach before a match can guide the decision of the position. If the coach is not open-minded, his viewpoint can be biased, ignoring other important characteristics of a player that the human eye may not perceive. Nowadays, the *modus operandi* in the clubs of the best football leagues is changing. The coach is supported in all his choices by the technical staff, which is also composed by match analysts and scouts. The professionals involved in match analysis and scouting roles work with a huge amount of data, to be properly interpreted. In particular, the match analyst tries to understand the tactical situation during the training sessions and matches, creating a summary for each player of the team. This type of analysis is conducted with proprietary data which cannot be accessed by the external environment, unless paid.
Given the interest to support the new technologies available to coaches and their technical staff, this thesis focuses on predicting the best position of a footballer in the pitch analyzing technical, mental and physical indicators of each player. The data used in the thesis are the data available for free from *FIFA*[G] *22* videogame. The goalkeeper position will not be considered, because prediction is focused on the outfield players that are involved in the dynamics of the football match. Moreover, a goalkeeper owns specific attributes and he cannot be judged outside the penalty area, even though there could be very rare exceptions like the Brazilian former-goalkeeper Rogério Ceni who scored 131 goals in his football career because of specialist of free kicks and penalties (see *Rogério Ceni*). Anyway, it represents an exception. The purpose of this thesis is not to replace the human eye, but rather to offer an objective support to the decision-making process regarding the position of a footballer in a football match in order to provide a data-driven perspective.

1.2 Related works

At time of writing, not much specific research has been done to predict the position of a footballer. Nevertheless, there are some interesting studies conducted on this field.
The thesis takes inspiration from the study in Bosu Babu et al. (2022), which focuses on predicting one of the fourteen available football player's position through Machine Learning techniques as Decision Tree, Random Forest, K-Nearest Neighbour, Naive Bayes and Support Vector Machine. Every model has been trained on a dataset that includes all the *FIFA 18* videogame footballers. Each footballer is associated to the standard 29 player's attributes and, to simplify the problem, the original response variable has been transformed into 14 differ-

ent binary attributes in order to obtain a single-class classification problem. Random Forest model turns out to be the best algorithm in terms of accuracy; moreover, 0.9 of accuracy is returned at the prediction stage.

Bazmara and Jafari (2013) use K-Nearest Neighbour classifier to investigate the quality of the players in order to predict one of the ten available footballers' positions, goalkeeper excluded. Data are taken from an experimental group of 150 footballers aged between 18 and 31, in which each footballer has physical, mental and technical attributes. Before training the model, the choice of K parameter and the distance type has been studied and. On average, the best model has a K value between 5 and 10 and distance type equal to cosine and correlation. After training the model, the accuracy obtained in the prediction phase was 0.97.

Bazmara (2014) suggests the implementation of a football player position identification system applying fuzzy logic approach, in which the knowledge is represented by if-then rules with *AND* logical operator inside the *if* conditions. There are ten possible players' positions to predict, goalkeeper excluded, and a number of attributes within the 24 available properties is fixed for each player position. After giving in input all the values of the properties for 264 players, an accuracy score is returned for each position and the sum of these scores is equal to 100. The fuzzy system returns a 91% accuracy value, suggesting that the performance is satisfactory.

Razali et al. (2017) use Machine Learning techniques as Bayesian Networks, Decision Trees and K-Nearest Neighbour to predict ten possible player's positions, goalkeeper excluded. Data are taken from the Football Player Information System at Bukit Jalil Sports School (BJSS), in which 100 young players have been rated from 1 to 10 for each physical, mental and technical attribute. The models have been trained and tested with *WEKA*[G] (Waikato Environment for Knowledge Analysis). The average accuracy is 0.98 and the framework suggest to be promising.

Zixue and Pan (2021) suggest the use of a back-propagation neural network to predict the best position of a player by distinguishing defender, midfielder and attacker. The data source is *Wyscout*, a famous football analysis platform which contains video data and specific information about the matches. Specific physical and technical characteristics have been selected after having conducted an analysis of the variance. After that, cross-validation has been applied for finding the best hyperparameters of the model, and the model has been trained on the complete dataset. The prediction phase returns good results for defenders and midfielders, but low results for attackers due to unbalanced proportion of observation between the different positions.

The above mentioned papers carried out analyses using only Machine Learning techniques. The thesis intends to exploit previous applications of Machine Learning techniques to the dataset constructed from the *FIFA 22* videogame, but also to extend the analysis including Data Mining approaches, in order to investigate the relationship between players' positions and players' features more deeply.

1.3 Programming languages and tools

The thesis work has been conducted with a laptop which holds the following specifications:

- *OS*[G] (Operating System): Microsoft Windows 10 Home 64-bit, 22H2 version (build OS 19045.2364);
- CPU: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50GHz;
- RAM: 16.00 GB (15.80 GB usable);
- GPU (integrated): Intel(R) Iris(R) Plus Graphics;
- GPU (dedicated): NVIDIA GeForce GTX 1050 with Max-Q Design.

Unfortunately, the dedicated GPU has never been employed because the programming language packages used only allow to work with the supplied CPU.

1.3.1 Programming languages

R (R Core Team, 2022) and Python (Van Rossum and Drake, 2009) are the programming languages used in this thesis. More in detail, *R* 4.2.1 version and Python 3.11 64-bit version have been used. Below there are all the details about these two programming languages.

- **R** is a simple programming language for statistical computing, data manipulation and analysis developed and maintained by the R Core Team and the R Foundation for Statistical Computing. *R* includes conditionals, loops and user-defined functions. Moreover, it allows to manage data through arrays and matrices and to easily handle and store data. Then, it provides a large amount of packages for statistical techniques and data visualization. Finally, C and C++ programming languages code can be linked to the *R* code and executed for advanced tasks. In this thesis, *R* has been used for the implementation of the Data Mining techniques. *R* code is available in Appendix A.

- **Python** is a high-level and object-oriented programming language developed and maintained by Python Software Foundation. Python is an interpreted language, so it works in whatever OS in which a Python interpreter is installed. Dynamic typing and dynamic binding are important properties that Python holds. It is suitable for the development of short programs and all the constructs used in more developed programming languages (as C++ or Java) are available and they can be used in a simpler and faster way. Its syntax is quite easy and it does not require too much effort to be learned. Finally, it provides a large amount of libraries, from the mathematical ones to the most recent Artificial Intelligence libraries. In this thesis, Python has been used for the dataset creation and the implementation of the Machine Learning techniques. Python code is available in Appendix B.

1.3.2 Tools

RStudio and Visual Studio Code are the tools used in this thesis. More in detail, RStudio 2022.12.0 Build 353 version and Visual Studio Code 1.74.1 version have been used. Below there are all the details about these two tools.

- **RStudio** is a free and open-source *IDE*[G] (Integrated Development Environment) for *R* programming language. It includes a console and tools for plotting graphs, history, debugging and workspace management. Moreover, it allows to execute chunks of code. The RStudio executable is distributed among the most spread OS as Windows, Linux and macOS. RStudio has been used for the implementation of Data Mining techniques.

- **Visual Studio Code** is a source code editor developed and maintained by Microsoft. It includes debugging tools, syntax highlighting, automatic code completion, snippets, code refactoring, and embedded Git. Visual Studio Code functionalities can be extended through the installation of extensions, which allows to add advanced editor tools and to provide additional support for programming languages. Visual Studio Code can be used for a wide range of programming languages, Python included. It is available for the most spread OS as Windows, Linux and macOS. Visual Studio Code has been used for the implementation of machine Learning techniques.

1.4 Personal motivations

My study plan of Master's degree has been focused on Artificial Intelligence, learning from a theoretical point of view in-depth analysis of data and how data allow to create intelligent systems. I was therefore eager to merge my studies with my greatest passion that I have since I was a child, namely, football. At time of writing, football analytics is under-explored compared to other sports such as basketball or ice hockey, but I sensed that interest in this field is growing more and more every day. For this reason, I decided to give my little contribute on predicting the best position that a footballer can take on a football pitch. My wish is that my dissertation could help the creation of an automatic tool which supports the choices of the coach and his staff in professional and non-professional clubs.

1.5 Thesis structure

- Chapter §1 is the current chapter.
- Chapter §2 describes the dataset, analyzing in depth every attribute. The way in which the dataset has been created from scratch is explained at the end of the chapter.
- Chapter §3 focuses on analyzing the attributes of the dataset from a graphical point of view. From this type of analysis, theoretical relationships between the response variable and all the attributes of the dataset can be inferred.
- Chapter §4 describes the Data Mining techniques, that will be applied for discovering the most significant relationships, from a theoretical point of view.
- Chapter §5 explains the final results obtained from each Data Mining technique.
- Chapter §6 describes the Machine Learning techniques, that will be applied for making predictions, from a theoretical point of view.
- Chapter §7 explains the final results obtained from each Machine Learning technique.
- Chapter §8 shows the performances about the best algorithms found applying a reduction of the response variable classes.
- Chapter §9 gives a final summary about the comparison between the Data Mining techniques and the Machine Learning techniques. Results are compared considering the metrics values and the training execution times for each algorithm.

2
The FIFA dataset

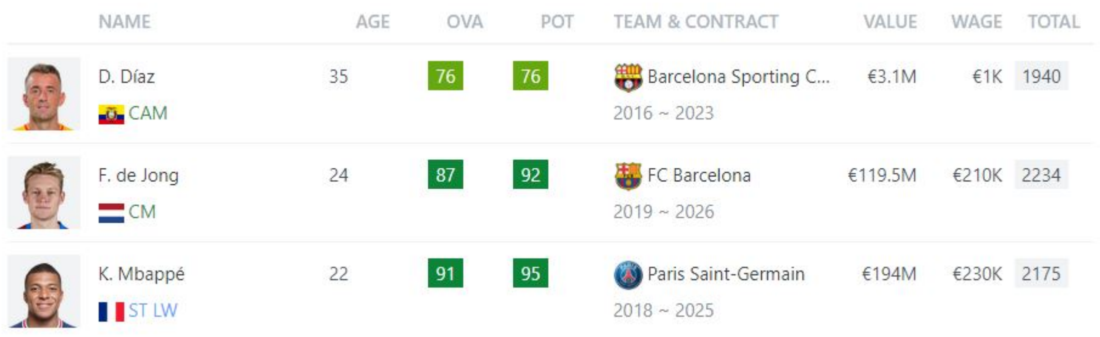
In order to solve the task of predicting the most suitable position for a player in a football pitch, it is necessary to look for a coherent dataset with the football domain. At the time of writing, there are plenty of sources which are available on demand through the payment of a specific fee, as the notorious *OPTA* and *Wyscout*. Nevertheless, this type of data is not meaningful for our task because the focus is on the football matches and the statistics related to them. So, it has been opted for gathering players' information from the most famous football video game named *FIFA 22*. The reasons are the following:

- data are about the characteristics of every player, both from the technical point of view and from the physical and mental ones;
- data can be obtained for free.

Surfing on the Web, there are some dataset available about the past version of FIFA videogame. Anyway, it has been decided to obtain data with a *Python*[G] program which *"scrapes"* the last version of the game at the end of August, in order to have the most up-to-date data possible. Trivially, it is a matter of fact that skills decrease over the years due to physical deterioration. Therefore, it would be useless to predict the actual best position for a player using old data, as it would not be realistic. A good example can be made with one of the greatest footballer of all time like Lionel Messi. Indeed, in the first stage of its career at Barcelona, he was forced to play on the right wing because the other two forward positions of the 4-3-3 lineup were busy, but it was good as positioning because his main feature was dribbling. When the new coach Pep Guardiola came, Messi exploited as better as possible his talent with the new position assigned that was *False 9*. In the past years on his career (both with Barcelona and Paris Saint-Germain), he moved further back in the pitch, with less focus on assuming the Central Forward role getting on the end of moves in the box, and more focus on progressive plays and shooting from deeper positions. This position change depends from two factors: the different tactical arrangement of his teammates and the increasing effort in attacking due to age (see *Messi Evolution*). This is why it has been decided to create from scratch an up-to-date dataset.

2.1 The source of data: SoFIFA.com

All the information about FIFA video game football players are freely available from *SoFIFA.-com* (https://sofifa.com/), a discussion and stats platform for *EA Sports*[G]' video game series in which all the information about players are gathered every year since 2007. At the time of writing, all the players of *FIFA 22* videogame have been considered, with release date *August 18, 2022*.



NAME	AGE	OVA	POT	TEAM & CONTRACT	VALUE	WAGE	TOTAL
D. Díaz CAM	35	76	76	Barcelona Sporting C... 2016 ~ 2023	€3.1M	€1K	1940
F. de Jong CM	24	87	92	FC Barcelona 2019 ~ 2026	€119.5M	€210K	2234
K. Mbappé ST LW	22	91	95	Paris Saint-Germain 2018 ~ 2025	€194M	€230K	2175

Figure 2.1: How footballers appear in SoFIFA.com.
Source: https://sofifa.com/

Every web page contains sixty players (the first three player are viewed as in Figure 2.1), in which every player is provided by personal data and a fixed set of attributes called *"Player Attributes"*. More in detail, the latter are the data and information that determine the quality and the feature of a player's technical skills, behaviours and performance on the pitch (see *FIFPlay Player Attributes*). Every attribute is an integer value which goes from 0 to 99. The higher the value, the better the quality for a specific attribute.

0-39
40-49
50-59
60-69
70-79
80-89
90-99

Figure 2.2: All the values range to determine the quality of an attribute.
Source: https://www.fifplay.com/encyclopedia/player-attributes/

FIFA has determined a fixed number of values range which allows to understand the type of quality of a footballer for a specific attribute, as demonstrated in Figure 2.2. The quality of a player is

- **Very Poor** if the values range goes from 0 to 39;
- **Poor** if the values range goes from 40 to 49;
- **Fair** if the values range goes from 50 to 69;
- **Good** if the values range goes from 70 to 79;
- **Very Good** if the values range goes from 80 to 89;
- **Excellent** if the values range goes from 90 to 99.

At this point, everyone can wonder how the attributes values are assigned to every player. Well, this work is made by an EA Sports team led by Mr. Michael Müller-Möhring (nicknamed *"Triple M"*), the Head of Data Collection and Licensing at EA Sports. His task is to ensure players are given their attributes in the game and appear as lifelike as possible (see *Michael Muller-Mohring*). Mr. Michael Müller-Möhring works with a team of twenty-five EA Sports Producers and four hundred outside data contributors, even accompanied by a community of over six-thousands FIFA Data Reviewers (a.k.a. Talent Scouts). The latter are volunteers who come from all over the world and in their life they are coaches, scouts or trivially football fans. This community is so important because it is unfeasible for EA Sports' staff to watch every single player in every single game, so the FIFA Data Reviewers help to maintain and update the attributes, the ratings and the compositions of a team in game (see *FIFA player ratings*). A relevant issue is the subjectivity in assigning value to a specific attribute due to internal biases. How to face the reliability problem of the FIFA dataset? Its reliability is guaranteed maintaining the attributes values fairly constant; indeed, attributes values must reflect the current general ability of the player and should not go up and down when the player has just one good or bad week (see *Football Talent Scout*). The more drastically the values change, the more likely the values are assigned in a subjective way. So, the attributes values are reviewed with attention by the main EA Sports team and only after verification they decide if to slightly modify the values in the next FIFA update or not.

2.2 List of FIFA players' personal data

It seems obvious, but before knowing what are the players' attributes it is important to underline what are the personal data which every player in FIFA owns. These ones are related to personal data, in a similar way to the information written in an identity card.
The personal data are listed below.

- **Name**: The name of the player, which is indicated with the initial letter of the name, followed by a dot plus a space and followed by the entire surname (i.e., Alberto Gobbo becomes A. Gobbo).
- **Age**: The age of the player.
- **Height**: The height of the player measured in centimeters.
- **Weight**: The weight of the player measured in kilograms.
- **Preferred Foot**: The foot the player uses more frequently and it can assume two values, that are *Left* or *Right*.
- **Best Position**: The most suitable position the player would like to play in a football pitch and it can assume a certain number of values that are indicated in the Section § 2.5.

The reported data are only the subset of the basic data available in *SoFIFA.com* (https://sofifa.com/) chosen during the process of the dataset creation (Section § 2.6), because not all personal data are significant to be known. Only *Height*, *Weight* and *Best Position* personal data will be analyzed in Chapter § 3.

2.3 List of FIFA players' attributes

Before starting to analyze how the dataset has been created, it is so important to know what are the *FIFA 22* player attributes. Notice there are two different list of attributes:

- one for outfield players (Subsection § 2.3.1);

- one for the goalkeeper (Subsection § 2.3.2).

All the *FIFA 22* attributes are responsible for determining a player's quality (see *FUT Player Attributes*) and they allow to understand strengths and weaknesses. They are in total *thirty-five*.

2.3.1 Outfield players' attributes

The outfield players' attributes are exactly *twenty-nine* and they are personal data and information which determine a player's quality and feature on the field. These features concerns the abilities, the skills and the performance of the player.
The outfield players' attributes are listed below.

1. **Acceleration**: The increment of a player's running speed. It is related with Sprint Speed attribute and the higher the Acceleration value the shorter the time needed to reach Sprint Speed.

2. **Aggression**: The frequency and the aggression level of tackling an opposing player, even determining the player's willpower or diligence during a match. It is related with Jumping, Strength, Sliding Tackle and Standing Tackle attributes.

3. **Agility**: How rapid and refined a player is when manages the ball. It is related with Dribbling attribute.

4. **Balance**: The ability to stay upright and stable during a physical challenge. It is related with Dribbling, Acceleration and Sprint Speed attribute.

5. **Ball Control**: The ability of a player to manage the ball both when he receives it and when the ball is stuck to his feet. It is related to Dribbling attribute and the higher the Ball Control value the less likely the player sweep the ball away while he is managing it.

6. **Composure**: The state of a player when he starts to feel under pressure, so how lucid is in frustrating scenarios.

7. **Crossing**: The accuracy of a player when crosses the ball into the opponent area both stationary and in motion.

8. **Curve**: The ability of a player to give a curve effect to the ball both when passing and when shooting.

9. **Defensive Awareness**: a.k.a. *Marking*, the ability to defend and watch over an opposing player in such a way as to stop opposing ball possession.

10. **Dribbling**: The ability of a player to take the ball and overcome an opponent while he has the ball control. It is related to Ball Control attribute.

11. **Finishing**: The ability of a player to score using his feet inside the opponent area. It has not related with Heading Accuracy and Long Shots attributes.

12. **Free Kick Accuracy**: The accuracy of a player to score free kicks. It is related with Curve attribute.

13. **Heading Accuracy**: The accuracy of a player when hitting the ball with the head both for passes and shots.

14. **Interceptions**: The ability of a player to intercept opponent passes.

15. **Jumping**: The ability of a player to jump off the ground by hitting the ball with the head. It is related with Strength, Aggression, Heading Accuracy and Height attributes.

16. **Long Passing**: The accuracy of a player to perform an aerial long pass to his teammate. Moreover, the higher the Long Passing score, the faster the ball is.

17. **Long Shots**: The accuracy of a player to score using his feet outside the opponent area.

18. **Penalties**: The accuracy of a player to score the penalty kicks.

19. **Positioning**: The ability of a player to catch the right position on the pitch during a game.

20. **Reactions**: How rapidly a player reacts to what happens in the pitch. It is not related with Sprint Speed, on the other hand it is related with Dribbling and Sliding Tackle.

21. **Short Passing**: The accuracy of a player when performs a short non-aerial pass to his teammate. Moreover, the higher the Short Passing value, the faster the ball is.

22. **Shot Power**: The power which a player impress when hitting the ball for a shot on target. Moreover, the higher the Shot Power value, the faster and the more distant the ball is.

23. **Sliding Tackle**: The ability of a player to tackle the opponent player using his legs, taking away the ball to the opponent himself and trying to not commit fouls. It is related with Standing Tackle attribute.

24. **Sprint Speed**: The top speed which a player achieves while runs.

25. **Stamina**: The ability of a player to hold up physical and mental efforts during a game. The higher the Stamina value, the less likely the player is to get injured easily. Moreover, it determines how rapidly a player recovers for the next match. Finally, it is related with Sprint Speed attribute; indeed, Stamina determines how long a player can sprint before slowing down and get tired.

26. **Standing Tackle**: The ability of a player to tackle the opponent player while standing, taking away the ball to the opponent himself and trying to not commit fouls. It is related with Sliding Tackle attribute.

27. **Strength**: How strong a player is physically. The higher the Strength value, the more likely a player win a physical challenge.

28. **Vision**: The awareness of a player about the position of his teammates and opponents around him. It is related with Long Passing attribute; indeed, the higher the Vision value, the more likely to execute a right long pass to the teammates.

29. **Volleys**: The accuracy and the strength of a player while executing a volley shot on target. It is related with Balance attribute.

Interestingly, the attributes are classified in different ways in the recent version of FIFA video game. The most common type of classification concerns the division into *physical*, *mental* and *technical*. The attributes are divided as follows (Table 2.1):

Table 2.1: Attributes divided in physical, mental and technical classes.

Physical	Mental	Technical
Acceleration	Aggression	Ball Control
Agility	Positioning	Crossing
Balance	Composure	Curve
Jumping	Interceptions	Defensive Awareness
Reactions	Vision	Dribbling
Sprint Speed		Free Kick Accuracy
Stamina		Finishing
Strength		Heading Accuracy
		Long Passing
		Long Shots
		Penalties

		Short Passing
		Shot Power
		Sliding Tackle
		Standing Tackle
		Volleys

Another recent type of classification concerns the division of the attributes in six classes which is very common in *FUT*[G] (FIFA Ultimate Team), an online game mode that lets a video gamer to build his dream squad (see *FIFA 22 Ultimate Team*). Each attribute indicates the player skill level and these attributes are the following (see *FIFA 22 Player Attributes*):

- **DRI**, a.k.a. *Dribbling*, notes ball control, agility and balance;
- **DEF**, a.k.a. *Defence*, notes tackling and marking;
- **PHY**, a.k.a. *Physical*, notes strength and stamina;
- **PAC**, a.k.a. *Pace*, notes the speed and the acceleration of the player;
- **PAS**, a.k.a. *Passing*, notes ability to successfully pass the ball with vision;
- **SHO**, a.k.a. *Shooting*, determines finishing skill and shot power.

Moreover, the latter attributes are classified as follows (see *FIFA 22 Player Attributes*):

- the attributes *DRI*, *DEF* and *PHY* are useful to define how well a player controls the ball;
- the attributes *PAC* and *PAS* are useful to define how well a player moves up the pitch;
- the attribute *SHO* is useful to define how well a player scores.

So, the twenty-nine attributes are divided as follows (Table 2.2):

Table 2.2: Attributes divided in DRI, DEF, PHY, PAC, PAS and SHO classes.

DRI	DEF	PHY	PAC	PAS	SHO
Agility	Intercep-tions	Jumping	Sprint Speed	Vision	Finishing

Balance	Heading Accuracy	Stamina	Acceleration	Crossing	Positioning
Reactions	Defensive Awareness	Strength		Free Kick Accuracy	Shot Power
Composure	Standing Tackle	Aggression		Long Passing	Long Shots
Ball Control	Sliding Tackle			Short Passing	Penalties
Dribbling				Curve	Volleys

So, we can conclude the twenty-nine attributes are responsible on defining the values of these six attributes. The values of the latter are also available from *SoFIFA.com* (https://sofifa.com/), so there is no need to calculate them.
A graphical analysis about the relationship between the attributes of a specific class, in order to understand what relationship could be useful for predictive models, is available in the chapter § 3.

2.3.2 Goalkeeper's attributes

The goalkeeper's attributes are exactly *six* available for every player, even though the player itself plays another role outside. Indeed, if during a football match substitutions are over and suddenly the goalkeeper gets injured or is sent off, an outfield player will have to fill the vacant role. So, even if it is not his role, one outfield player will be chosen by the coach in emergency cases being able to rely on his characteristics as a goalkeeper.
The goalkeeper's attributes are listed below.

1. **Diving**: The ability of a goalkeeper to make a save while diving into the air. It is related with Height attribute.

2. **Handling**: How capable is the goalkeeper to catch the ball and keep it.

3. **Kicking**: How accurate and long is a goal kick of a goalkeeper, both with the ball on the ground and in the air.

4. **Positioning**: The ability of a goalkeeper to hold the right position both when saving whatever type of shot and when tries to intercept a cross.

5. **Reflexes**: How agile a goalkeeper is when he makes a save. The higher the value, the faster the goalkeeper reacts to a shot.

6. **Speed**: The ability of a goalkeeper to close an opponent in one-on-one situations.

2.4 List of FIFA players' special attributes

The source web page provides other attributes that need to be considered as *special* because they could contribute on the players quality improvement. The special attributes are listed below.

- **Weak Foot**: The shot power and ball control for the weaker foot of the player than his preferred foot. It is rated from 1 to 5. The higher the rate, the higher shot power and ball control have the player. For instance, a weak foot rated as 3 of 5 means it is slightly above the average in terms of ball controlling and shot power (see *Weak Foot*).

- **Attacking Work Rate**: The rate of a player's behavior on the pitch in terms of attacking work. It defines the effort of a player to participate in attacks even when he is out of position (see *Work Rate*). It can assume the following values (see *FIFA 22 Work Rates*):

  - *Low*, which means a player will not play much further than from outside the defensive penalty area up to the defensive midfield;
  - *Medium*, which means a player will play from the midfield to outside the penalty area;
  - *High*, which means a player will push deep into the attacking third, into the box and into the corners of the wings.

- **Defensive Work Rate**: The rate of a player's behavior on the pitch in terms of defensive work. It defines the effort of a player to participate in defenses even when he is out of position (see *Work Rate*). It can assume the following values (see *FIFA 22 Work Rates*):

  - *Low*, which means a player will not drop much further than outside the attacking penalty area up to the attacking midfield;
  - *Medium*, which means a player will play from the midfield to outside the defensive penalty area;

  - *High*, which means a player will drop deep into the defensive third, into the box, into the corners of the wings.

- **Body Type**: The body composition of a player even combined with its height. According to the attributes available in *SoFIFA.com* (https://sofifa.com/), the values available for this attribute are the following: *Lean (170-185)*, *Normal (170-185)*, *Stocky (170-185)*, *Lean (185+)*, *Normal (185+)*, *Stocky (185+)*, *Lean (170-)*, *Normal (170-)*, *Stocky (170-)* and *Unique*.

2.5 List of FIFA players' positions

This section is focused on the specific personal FIFA player attribute named *Best Position* which can assume a fixed set of values. This attribute plays the role of response variable which will be predicted by the previously listed information. As a preliminary step, details about roles in a football pitch according to FIFA are reported.
During his career, a footballer player could assume more than one position in the football pitch according to the coach choices, the composition of a team and the player's skills which can alter going over with age. Anyway, a player will have always a preference, and so a best position on the field.
The position of players on the field determines their roles and assignments. In football game, every team must play with 11 players at the start of the game, so the coach is responsible to define the lineup and to assign a particular position on the pitch. More in detail, a team is shaped up of one goalkeeper and ten outfield players who fill various defensive, midfield, and attacking positions (see *Position*). The number of players in these positions depends on the composition of a team. For example, the 3–5–2 lineup is composed by three defenders, five midfielders and two attackers (see *Formation*). In *FIFA 22* video game, there are sixteen player positions which can be assigned to outfield players based on his abilities and skills. Even considering the goalkeeper position, the total number of positions to be considered is seventeen. The football game is dynamic and consequentially player positions can be dynamically changed, for example, when tactics or composition of a team are changed.
The positions of a football player are shown in Figure 2.3. The list of *FIFA 22* positions is reported below:

- **GK**, a.k.a. *Goalkeeper*, is the player defending his own goal with the main aim to prevent goals from the opposing team. He is the only player who can use his hands within his penalty area. Moreover, he wears a different colored kit than his teammates and gloves to protect himself from injury (see *Goalkeeper*);

- **CB**, a.k.a. *Centre Back* or central defender or centre-half, is a defender positioned in front of the goal and near his area with the main aim to prevent opposing players from scoring. He is led to tackle the opponents, intercepting shots and passes, to contest headers and to mark attackers in order to apply pressing on them and prevent the ball from being received (see *Centre Back*);

- **RB**, a.k.a. *Right-back* or right full-back, is a defender who occupies the wide positions, in particular the right side of the defensive line. He is preferably right-footed (see *Right Back*);

- **LB**, a.k.a. *Left-back* or left full-back, is a defender who occupies the wide positions, in particular the left side of the defensive line. He is preferably left-footed (see *Left Back*);

- **RWB**, a.k.a. *Right Wing Back*, is a wing-back defender who occupies the right side of the field. He is led to defend and attack with high stamina, together with a good ability on crossing. He is preferably right-footed (see *Right Wing Back*);

- **LWB**, a.k.a. *Left Wing Back*, is a wing-back defender who occupies the left side of the field. He is led to defend and attack with high stamina, together with a good ability on crossing. He is preferably left-footed (see *Left Wing Back*);

- **CDM**, a.k.a. *Central Defensive Midfielder*, is a central midfielder with defensive roles with the aim to help his defenders positioning in front of them or marking a specific opponent player. A good CDM has high values in Interceptions, Heading Accuracy, Defensive Awareness and tackling attributes (see *Central Defensive Midfielder*). There are two types of central defensive midfielder, which are:

  - *Holding midfielder*, who is positioned near to his defenders in many game situations. He is usually physically strong;
  - *Deep-lying Play-maker*, who is a holding midfielder with higher abilities in passing the ball than tackling, even having a good vision and good capabilities in maintaining the ball possession;

- **CM**, a.k.a. *Central Midfielder*, is a player positioned between defence and attack with the aim to dominate the game in the centre of the pitch. Moreover, he is led to pass the ball to his attacking midfielders and forwards supporting the attack phase and he can try to score with long shots (see *Central Midfielder*). There are two types of central midfielder, namely:

  - *Box-to-box midfielder*, who has good skills and capabilities to perform well in whatever game phase;
  - *Mezzala*, who is a half-winger that covers one of the two central midfielders position when the composition of a team provides two midfielders in the middle of the field. He supports the other central midfielder and the central attacking midfielder;

- **RM**, a.k.a. *Right Midfielder*, is a midfielder who occupies the wide positions, in particular positioned on the right side and closer to the touchlines of the pitch. He has the same aim of a midfielder, with the addition of the ability to cross the ball into the opponent's penalty area to create scoring chances for his teammates (see *Right Midfielder*). He is usually right-footed;

- **LM**, a.k.a. *Left Midfielder*, is a midfielder who occupies the wide positions, in particular positioned on the left side and closer to the touchlines of the pitch. He has the same aim of a midfielder, with the addition of the ability to cross the ball into the opponent's penalty area to create scoring chances for his teammates (see *Left Midfielder*). He is usually left-footed;

- **CAM**, a.k.a. *Central Attacking Midfielder*, is an advanced central midfielder, usually positioned between central midfielders and forwards. He is led for offensive tasks, with the aim to help his forwards on scoring goals. A good CAM has high values in Dribbling, Agility, Balance, Ball Control, Positioning, Acceleration, Shooting and Finishing attributes (see *Central Attacking Midfielder*). There are three types of central attacking midfielder, namely:

  - *Advanced play-maker*, a technical player with capabilities in passing the ball and Dribbling. Moreover, he has a good vision and he delivers passes to his strikers putting the opposing defense in difficulty;
  - *False Attacking Midfielder*, who is able to draw opposing players out of position and to create space for his teammates in attacking phase. Moreover, he is usually creative and tactically intelligent with good abilities in vision, passing the ball and technique;
  - *False 10* or Central Winger, who tries to move out of his position and carry the ball to help the wingers to get up in the side bands of the pitch. He is usually good abilities in dribbling, ball control, shooting, vision and speed;

- **CF**, a.k.a. *Central Forward*, is a forward positioned at the center of the attacking line with the aim of attacking and scoring goals. He is able to move in a good manner and usually receives passes to score goals or makes an assist to his teammates. He starts behind the striker if the latter is present in the composition of a team. He can be of one type named *Target Man*, a central forward with the aim to win aerial balls both for scoring goal and creating goal chances for his teammates (see *Centre Forward*);

- **RF**, a.k.a. *Right Forward*, is an inside forward positioned on the nearest right side to the opposing team's area whose his aim is to score goals for his team. A good RF has high value in Ball Control (see *Right Forward*);

- **LF**, a.k.a. *Left Forward*, is an inside forward positioned on the nearest left side to the opposing team's area whose his aim is to score goals for his team. A good LF has high value in Ball Control (see *Left Forward*);

- **RW**, a.k.a. *Right Winger*, is an attacking player positioned in a wide position near the touchlines at the right side of the pitch. He is typically right-footed or two-footed and his aim is to overcome the opposing full-backs, to cross and to score goals. A good RW has high values in Ball Control, Dribbling and Speed (see *Right Winger*);

- **LW**, a.k.a. *Left Winger*, is an attacking player positioned in a wide position near the touchlines at the left side of the pitch. He is typically left-footed or two-footed and his aim is to overcome the opposing full-backs, to cross and to score goals. A good RW has high values in Ball Control, Dribbling and Speed (see *Left Winger*);

- **ST**, a.k.a. *Striker*, is a forward positioned in center of the attacking line with the aim of attacking and scoring goals. He is similar to a CF player, with the difference a ST player is able to distance from opposing defenders and to run into space for receiving balls from his teammates and score. A good ST has high values in Speed, Ball Control and Dribbling (see *Striker*).

These positions can be categorized into four different macro-roles which depend on the geographical position of the player in the composition of a team, namely:

- **Goalkeeper**, which includes only **GK** position;

- **Defender**, which includes **CB**, **RB**, **LB**, **RWB** and **LWB** positions;

- **Midfielder**, which includes **CDM**, **CM**, **RM**, **LM** and **CAM** positions;

- **Forward**, which includes **CF**, **RF**, **LF**, **RW**, **LW** and **ST** positions.

It is important to underline there are no players in *SoFIFA.com* who have **RF** and **LF** as forward best position. Due to this lack, the number of player's positions to consider from this point will be *fifteen*.

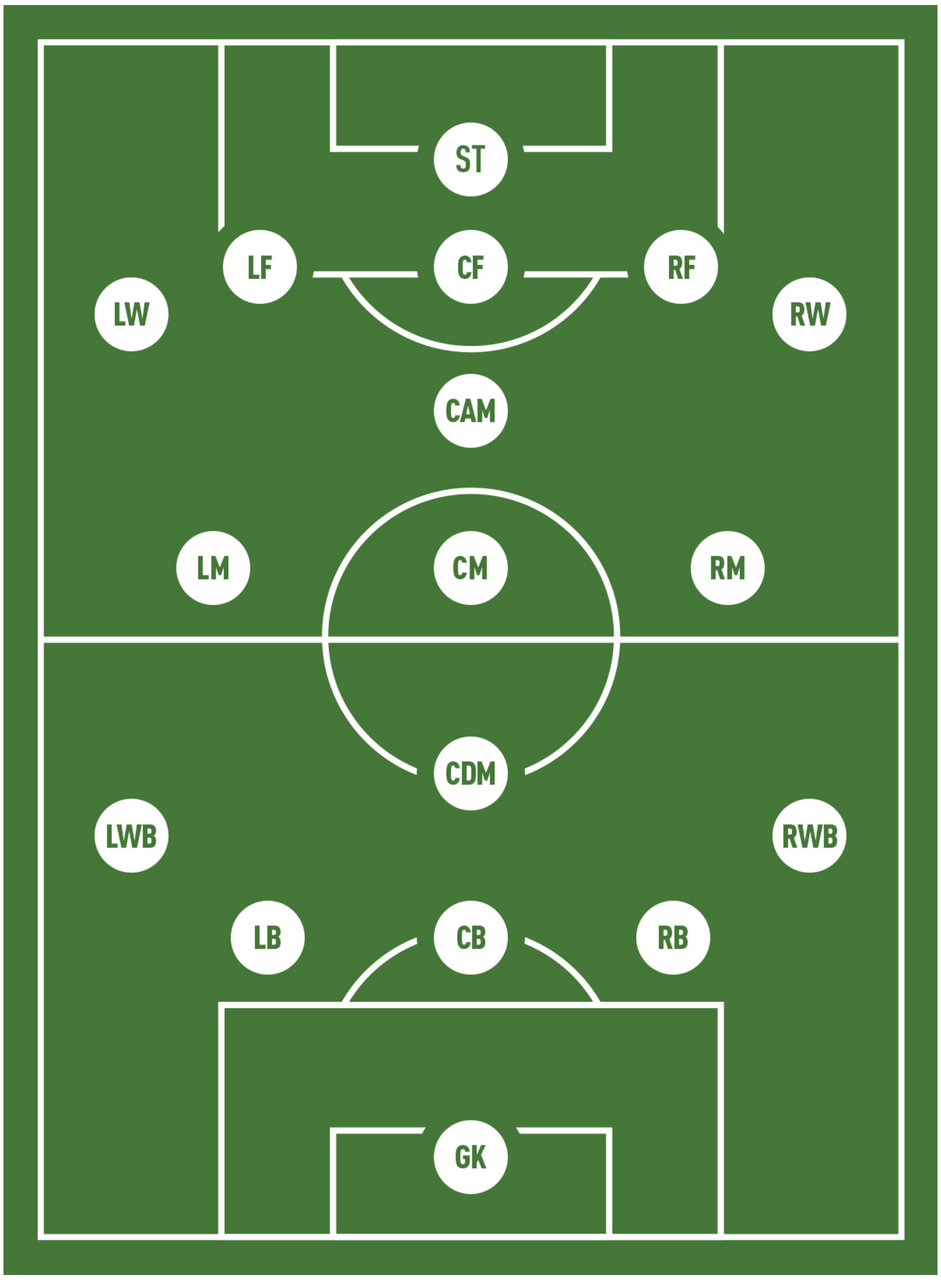


Figure 2.3: All the positions in a FIFA football pitch.
Source: https://www.fifplay.com/encyclopedia/position/

2.6 Creating the dataset

Now that all the attributes will be encountered are revealed, it is time to understand the process which has been applied for creating the dataset. This step is necessary to have up-to-date information about players and to have a full management of the attributes. Unfortunately, *SoFIFA.com* (https://sofifa.com/) does not provide a complete dataset to be downloaded. So, it is necessary to find a way to extrapolate data from the web site and saving them locally. This technique takes the name of *Web Scraping*[G], which allows to collect data from the Internet and parsing them into meaningful form (see *Web Scraping*). For this task, a program has been created which scrapes data from the already mentioned web site. The program has been written with Python (Van Rossum and Drake, 2009) programming language.
First, *BeautifulSoup* is the Python library used for extracting data from web out of *HTML*[G] and *XML*[G] files (see *Library BeautifulSoup*). After that, four different functions have been created and used in the following order:

1. **FirstPageWebScraping(url)** allows to extract the data from all the pages of the web page (except the first one), focusing on saving the players data. Parameter *url* is passed to the function, and it identifies the *URL*[G] of the web page from where it is applied the Web Scraping technique;

2. **NextPagesWebScraping(url, 60)** allows to extract the data from the first page of the web page, focusing on saving the attributes names and the first sixty players data sixty at a time. Parameters *url* and *60* are passed to the function, where the first identifies the URL of the web page from where it is applied the Web Scraping technique and the second is the offset used to change web page and directly inserted in the URL as last parameter;

3. **ManipulateAndCleanData()** allows to rectify and clean the data obtained after that *FirstPageWebScraping(url)* and *NextPagesWebScraping(url, 60)* functions have ended their job of data extraction. It returns *cleaned_dataset*, a cleaned dataset in *DataFrame* format which allows to obtain a two-dimensional tabular data structure (see *Data structure DataFrame*);

4. **GenerateCSVDataset(cleaned_dataset)** allows to take in input the *cleaned_dataset* returned by *ManipulateAndCleanData()* function and transform it in *CSV*[G] format.

Note that the URL that has been used is quite complicated with respect to the one written at the start of this chapter. Indeed, all the players' attributes available from the attributes

selection option have been added. Due to the excessive length of the URL, below all the attributes selected from *COLUMNS SELECTED* option are reported: Age, Height, Weight, Preferred Foot, Best Position, Crossing, Finishing, Heading Accuracy, Short Passing, Volleys, Dribbling, FK Accuracy, Long Passing, Ball Control, Acceleration, Sprint Speed, Agility, Reactions, Balance, Shot Power, Jumping, Stamina, Strength, Long Shots, Aggression, Interceptions, Positioning, Vision, Penalties, Composure, Marking, Standing Tackle, Sliding Tackle, GK Diving, GK Handling, GK Kicking, GK Positioning, GK Reflexes, Weak Foot, Attacking Work Rate, Defensive Work Rate, Body Type, Pace/Diving, Shooting/Handling, Passing/Kicking, Dribbling/Reflexes, Defending/Pace and Physical/Positioning.
All the implementation details will be explained below.

2.6.1 FirstPageWebScraping(url) method

The code concerning the current method is shown in Listing § B.1.
This method takes in input the URL (plus its final part "&offset=0") of the first web page. It verifies if the response of the request made with this URL is 200, which means that the server has replied with success to the client and giving the page to it (row 7 of Listing § B.1). If the response is negative an error is printed (row 15 of Listing § B.1), otherwise some operations are applied.
First, the entire HTML *page* source is caught and it is passed to *CatchWebPageTable(page)* (rows 7-8 of Listing § B.1), which is responsible to create a *BeautifulSoup* navigable object that allows to select the players' table only and to return it. In this case, it is mandatory to analyze the page source and to understand what are the HTML attributes which identifies the players' table.
Once the function returns *players_table*, it is passed to two different functions:

- *CatchFeaturesNames(players_table)*, which is responsible to select all the players' table rows with HTML tag *th* and finally to extract the values from each cell of each row and saving them in a specific list called *features_names_list*; in this case, the row in only one because the attributes name are on the top of the table. The code concerning the current method is shown in Listing § B.2;

- *CatchFeaturesValues(players_table)*, which is responsible to select all the players' table rows with HTML tag *tr* and finally to extract the values from each HTML tag *td* of each row and saving them in a specific list called *features_values_list*, which is finally converted in a *DataFrame* object called *results_features_values*; in this case, the row should be sixty as the number of players showed in every web page. The code concerning the current method is shown in Listing § B.3.

Finally, the *time.sleep(500)* function is called to suspend momentarily the execution of the program for five-hundred milliseconds (row 13 of Listing § B.1).

2.6.2 NextPagesWebScraping(url, 60) method

The code concerning the current method is shown in Listing § B.4. This method takes in input the URL (plus its final part "&offset=current_offset"of the next web pages. At the start, *current_offset* value is equal to 60, the parameter passed into the function which represents the initial offset (row 6 of Listing § B.4). It verifies if the response of the request made with this URL is 200, which means that the server has replied with success to the client and giving the page to it (row 8 of Listing § B.4). If the response is negative an error is printed (rows 17-20 of Listing § B.4), otherwise some operations are applied for a certain number of times until the offset is minor or equal of 19980 (the number of players available in FIFA).

First, the entire HTML *page* source is caught and it is passed to *CatchWebPageTable(page)*, which is responsible to create a *BeautifulSoup* navigable object that allows to select the players' table only and to return it (rows 9-10 of Listing § B.4). In this case, it is mandatory to analyze the page source and to understand what are the HTML attributes which identifies the players' table.

Once the method returns *players_table*, it is passed to the *CatchFeaturesValues(players_table)* method (Listing § B.3), which is responsible to select all the players' table rows with HTML tag *tr* and finally to extract the values from each HTML tag *td* of each row and saving them in a specific list called *features_values_list*, which is finally converted in a *DataFrame* object called *results_features_values*; in this case, the row should be sixty as the number of players showed in every web page. Notice that *CatchFeaturesNames(players_table)* method is not called because attributes names have been caught when the first web page has been scraped.

Finally, *current_offset* is updated summing 60 to its current value (row 14 of Listing § B.4) and the *time.sleep(500)* function is called to suspend momentarily the execution of the program for five-hundred milliseconds (row 15 of Listing § B.4).

2.6.3 ManipulateAndCleanData() method

The code concerning the current method is shown in Listing § B.5. This method is responsible for manipulating data in order to give to them a meaningful form. Moreover, it is necessary to check the character errors or null values, with the aim to return a dataset named *cleaned_dataset* which is as clean as possible.

First, it is necessary to split each row to cells using *split()* method (row 6 in Listing § B.5). In this way, every row contains a certain number of values which originally were separated by commas.

After that, a good practice is to set the dataset columns names with the values contained in *features_names_list* (row 4 in Listing § B.5). Additionally to it, some attributes need to be renamed, both to be aligned with the real names (as written in the Section § 2.3) and to be without space characters as if they were variables of a programming language (rows 15-46 in Listing § B.5). Then, some character errors have been noticed in some attribute values. For instance, *Height* and *Weight* values contained respectively the *"cm"* and *"kg"* wording, but for having a quantitative analysis it is better to have only numeric values without combined without any other character. This problem has been solved using the method *replace()* and applying specific regular expressions (rows 54-60 in Listing § B.5). Then, all the white-space from the beginning and at the end of the attribute values have been removed using the *strip()* function (rows 63-64 in Listing § B.5).

Finally, all the rows containing at least one empty cells or with null values have been removed (row 67-72 in Listing § B.5). This operation is necessary because a dataset needs to be uncorrupted to be analyzed; indeed, having missing values in a row corresponds to have an incomplete observation.

2.6.4 GenerateCSVDataset(cleaned_dataset) method

The code concerning the current method is shown in Listing § B.6. This method takes in input *cleaned_dataset* returned by the method in the Section § 2.6.3.

Only one operation is applied and concerns the *DataFrame* dataset conversion in a CSV format with encoding *UTF-8*[G] (row 2 of Listing § B.6).

2.6.5 Precautions for a safe web scraping

There are some aspects of a web service that needs to be considered for activities like Web Scraping.
The first precaution is to examine the *ToS*[G] (Terms of Service), which is a type of document stating details about what a service provider is responsible for as well as user obligations that must be adhered to for continuation of the service. Users that do not follow the rules specified in a ToS are subject to termination (see *Terms of Service*). For the specific case of this thesis, it is important to note any clauses surrounding accessing and republishing of data (Lewis and Wardrip-Fruin, 2010). As declared in its ToS, *SoFIFA.com "has a zero-tolerance policy regarding spam, pornography, copyright infringement, and abuse"* (*SoFIFA Terms of Service*). The data that has been extracted from it do not violate the copyright because the paternity of the data are always attributed to *SoFIFA.com* and there are only scientific purposes. Moreover, there is no possibility to transmit illegal material for criminal acts because it should be registered as member for the service provided by the website itself.
The last precaution is applied when downloading the HTML source page. Indeed, at reported in the Section § 2.6.1 and Section § 2.6.2, *"the time.sleep(500) function is called to suspend momentarily the execution of the program for five-hundred milliseconds"*. This operation is applied because, when downloading from a web service, it is considered customary for a crawler to wait for a couple of seconds between requests. Anything more than this may be flagged as an abuse and automatically banned by the web server (Lewis and Wardrip-Fruin, 2010). Anyway, it has been opted to reduce the time suspension of extracting data from the web site from two seconds to five-hundred milliseconds, in order to decrease the program execution time. Indeed, after some tests it was found the website does not ban even with a shorter time suspension.

2.7 Attributes names in the cleaned dataset

Attributes names listed in the Section § 2.2 and in the Section § 2.3 are slightly different from attributes names extracted from the *SoFIFA.com* source. Moreover, the latter have been modified in order to be similar to a variable name of a programming language as described in Section § 2.6.3. Table 2.3 includes the correspondence between the original attributes names and the ones used after the dataset cleaning.

Table 2.3: Name correspondence between original and cleaned dataset attributes.

Original dataset attributes	Cleaned dataset attributes
Name	Name
Age	Age
Height	Height_cm
Weight	Weight_kg
Preferred Foot	Preferred_Foot
Best Position	Best_Position
Acceleration	Acceleration
Aggression	Aggression
Agility	Agility
Balance	Balance
Ball Control	Ball_Control
Composure	Composure
Crossing	Crossing
Curve	Curve
Defensive Awareness	Defensive_Awareness
Dribbling	Dribbling
Finishing	Finishing
Free Kick Accuracy	Free_Kick_Accuracy
Heading Accuracy	Heading_Accuracy
Interceptions	Interceptions
Jumping	Jumping
Long Passing	Long_Passing
Long Shots	Long_Shots
Penalties	Penalties
Positioning	Positioning
Reactions	Reactions
Short Passing	Short_Passing
Shot Power	Shot_Power
Sliding Tackle	Sliding_Tackle

Sprint Speed	Sprint_Speed
Stamina	Stamina
Standing Tackle	Standing_Tackle
Strength	Strength
Vision	Vision
Volleys	Volleys
Diving	GK_Diving
Handling	GK_Handling
Kicking	GK_Kicking
Positioning	GK_Positioning
Reflexes	GK_Reflexes
Speed	/
Weak Foot	Weak_Foot
Attacking Work Rate	Attacking_Work_Rate
Defensive Work Rate	Defensive_Work_Rate
Body Type	Body_Type
DRI	Dribbling_Reflexes
DEF	Defending_Pace
PHY	Physical_Positioning
PAC	Pace_Diving
PAS	Passing_Kicking
SHO	Shooting_Handling

Character "/" means that there is no correspondence because the attribute is not available from the source web page. Notice that, starting at this point, the attributes names which will be used are the ones of the cleaned dataset.

3
Preliminary data analysis of FIFA dataset

This chapter focuses on analyzing FIFA dataset with graphical tools provided by the *R* (R Core Team, 2022) programming language. This step allows to identify patterns and trends on data, as well as potential relationships between the attributes and the response variable. Before visualization, a good practice is to pre-process data, in order to make them readable correctly without misunderstandings.

3.1 Pre-processing of dataset

The original dataset contains 19948 rows and 50 columns. The code concerning the dataset pre-processing is shown in Listing § A.1.
The first pre-processing step checks whether the dataset contains duplicates, in order to avoid data redundancy (row 1 of Listing § A.1). More in detail, duplicates are rows which are equals, with same values for every column. It has been found there are 318 duplicates on the original dataset. So, it is necessary to maintain only one row for every duplicate and remove the remaining ones. The dataset now contains 19130 rows and 50 columns.
The second step checks for the presence of missing values (row 2 of Listing § A.1). Many Machine Learning algorithms can fail if the dataset contains missing data or they can lead to a lack of precision in the statistical analysis. Our dataset contains no missing values.
Another step is to define the data type for every column which represents a player's attribute. *R* provides different data types, where *integer* and *factor* are the ones that are necessary for

dataset attributes. At the dataset creation moment, the standard data type was *character* for every attribute, so it is necessary a slight modification. First, all the attributes which assume values in a range from 0 to 99 are transformed into integers (row 4 of Listing § A.1), so quantitative covariates without the decimal part. Then, all the attributes with a limited set of string values are transformed into factors (rows 5-7 of Listing § A.1), so qualitative covariates with a fixed number of levels in alphabetical order. Notice that for factors *Best_Position*, *Attacking_ Work_Rate* and *Defensive_ Work_Rate* the levels order has been changed; indeed, for the first factor the order is changed to show players' positions going from the defensive roles to attacking ones, whereas for the remaining two factors the order is changed from low values to high ones.
The last step is to reduce the dimension of the dataset, by removing variables that do not offer useful information for the graphical analysis and the prediction activities (row 9 of Listing § A.1). Particularly, thirteen variables are not longer included:

- *Name* of the player;
- *Age* cannot influence directly the position of a player, because the latter depends on the players' attributes. It is a matter of fact that the higher the age the higher the likelihood that the physical condition has deteriorated, anyway, these type of data are already contained in other attributes and *Age* would result as redundant;
- *GK_Diving*, *GK_Handling*, *GK_Kicking*, *GK_Positioning* and *GK_Reflexes* refer to goalkeeper position, so they are useless for predicting whatever outfield player position;
- *Dribbling_Reflexes*, *Defending_Pace*, *Physical_Positioning*, *Pace_Diving*, *Passing_Kicking* and *Shooting_Handling* because their values strictly depend on the attributes described on Table 2.2 and they would result as redundant.

In addition, all the dataset rows which contain the goalkeeper value in *Best_Position* have been removed (row 10 of Listing § A.1) and the corresponding level has been removed from the covariate because empty (row 11 of Listing § A.1). Note that the goalkeeper attributes have been removed because the target of this thesis is to predict the best position for outfield players, since is not realistic goalkeepers do not change their position during their football career. Moreover, goalkeepers have specific attributes which have higher values with respect to outfield players' attributes, so it is very unlikely this position will be wrong in the prediction phase.

Normalization is not necessary for all *integer* type data because this technique is required only when features have different ranges. This is not the case, because every integer attribute goes from 0 to 99. At the end of this process, the dataset contains 17018 rows and 37 columns.

3.2 Graphical evaluation

At this point, it is possible to analyze the most important covariates which represent the players' attributes and the possible relationships between the ones in the dataset. The analysis has been conducted with *ggplot2 R* package (Wickham, 2016).

3.2.1 Response variable distribution

The response variable is *Best_Position*, which consists of 14 levels after the deletion of the goalkeeper level. It follows a multinomial distribution and a preliminary graphical inspection (Figure 3.1) suggests the levels are not homogeneous in terms of number of observations. Indeed, the most dominant level is CB with the 20,5% of the players who occupies this position, followed by ST with 17,4% and CAM with 16,3%. For what concerns the three least frequent levels, they all refer to forward positions and they are RW with 1,6%, LW with 1,0% and finally CF with 0,4%. Even grouping the levels according to the general outfield position, the number of observation for every level is unbalanced; indeed, the midfielders represent the majority of the players with 44,3%, followed by the defenders with 35,3% and finally followed by forwards with 20,4%. More in detail, on 17018 observations there are:

- 3497 observations for CB position;
- 974 observations for RB position;
- 906 observations for LB position;
- 326 observations for RWB position;
- 319 observations for LWB position;
- 1241 observations for CDM position;
- 979 observations for CM position;
- 1694 observations for RM position;
- 835 observations for LM position;

- 2772 observations for CAM position;
- 60 observations for CF position;
- 277 observations for RW position;
- 171 observations for LW position;
- 2967 observations for ST position.

While the most frequent positions are easily detected, the least frequent levels do not, because there could be an adequate number of observations to guarantee a satisfactory accuracy value in prediction phase. Even if the number of observations for the least frequent level CF should be quite satisfactory in relation to the total number of observations, it is necessary to pay attention to this phenomenon further on.

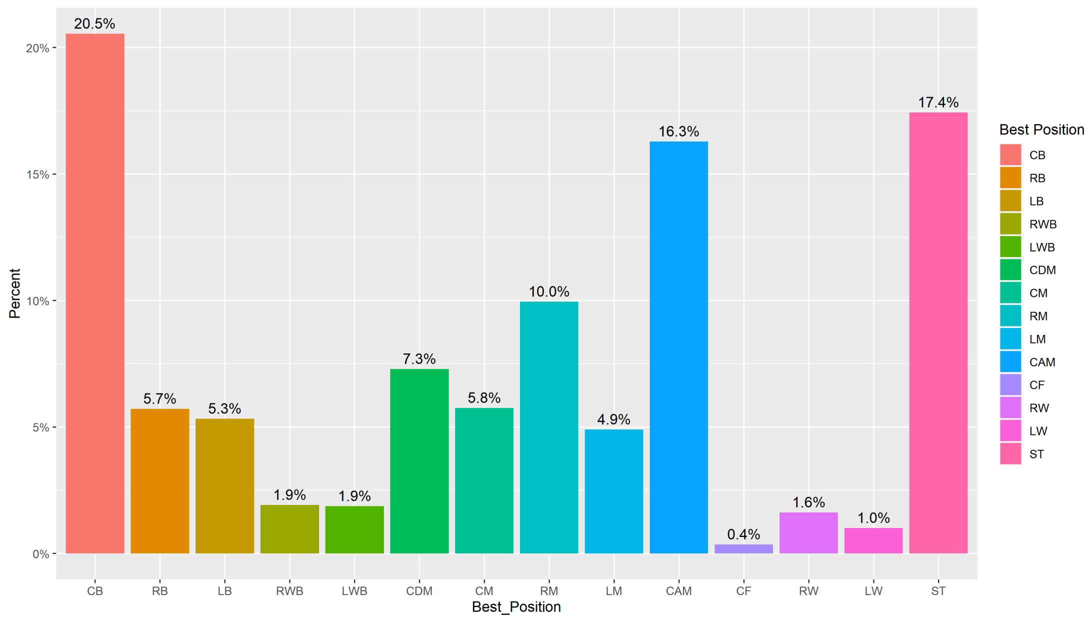


Figure 3.1: Distribution of the Best_Position response variable.

3.2.2 Response variable against categorical covariates

The relationship between the response variable and each categorical covariate is investigated using the *ggmosaic R* package. As shown in Figure 3.2, every rectangle area represents the proportion of cases for any given combination of levels. To understand if there could be a relationship, it is necessary to check if the conditional distributions of the two variables involved look similar. It can be supposed that there could be an important association between the response variable and the factors as *Preferred_Foot*, *Weak_Foot*, *Attacking_Work_Rate*, *Defensive_Work_Rate* and finally *Body_Type*. In each mosaic plot no conditional distribution is similar due to the different frequencies of the players' position with respect to the already mentioned factors. Moreover, it is possible to understand some statistical features. For instance, the comparison between the response variable and *Defensive_Work_Rate* in Figure 3.2 suggests that the rate decreases from players which occupy central defensive roles to players which occupy more offensive roles, as expected.

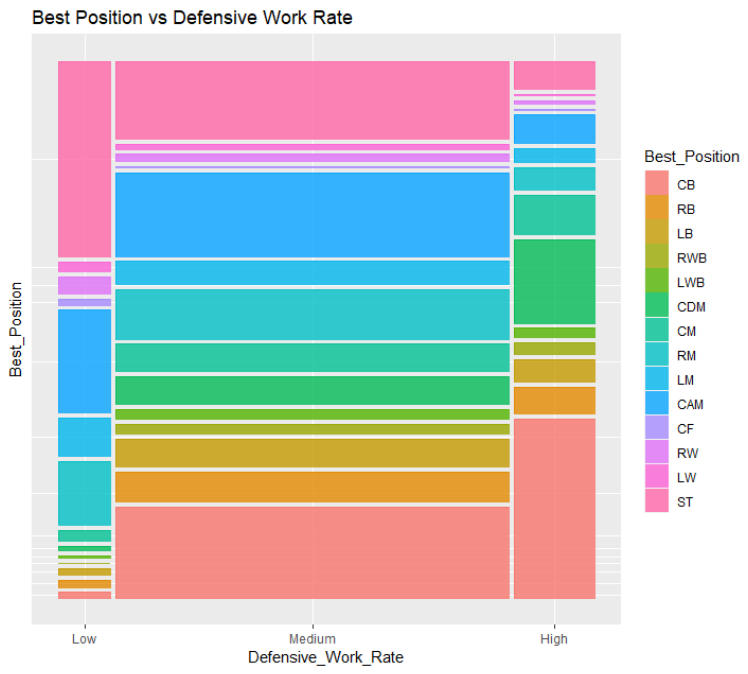


Figure 3.2: Mosaic plot of the response variable versus Defensive_Work_Rate.

3.2.3 Response variable against quantitative covariates

The relationship between the response variable and each quantitative covariate is investigated using the *boxplot R* package. As shown in Figure 3.3, it is displayed the center and the spread of a numeric variable in a format which allows to quickly understand the values range and comparing it to other covariates (Lantz, 2015). Due to the large number of quantitative covariates, only the most representative figures will be reported. For what concerns the relationship between the response variable and *Height_cm* in Figure 3.3, all the median values get around 180, except for CB and ST position in which they are slightly higher. Moreover, the majority of the boxplots tend to be quite symmetric, with many outliers for those positions that result to be the most frequent. The same behaviour is verified with *Weight_kg* covariate, but with different median values. For this reason, it can be suggested a relation between the response variable and the two latter quantitative variables.

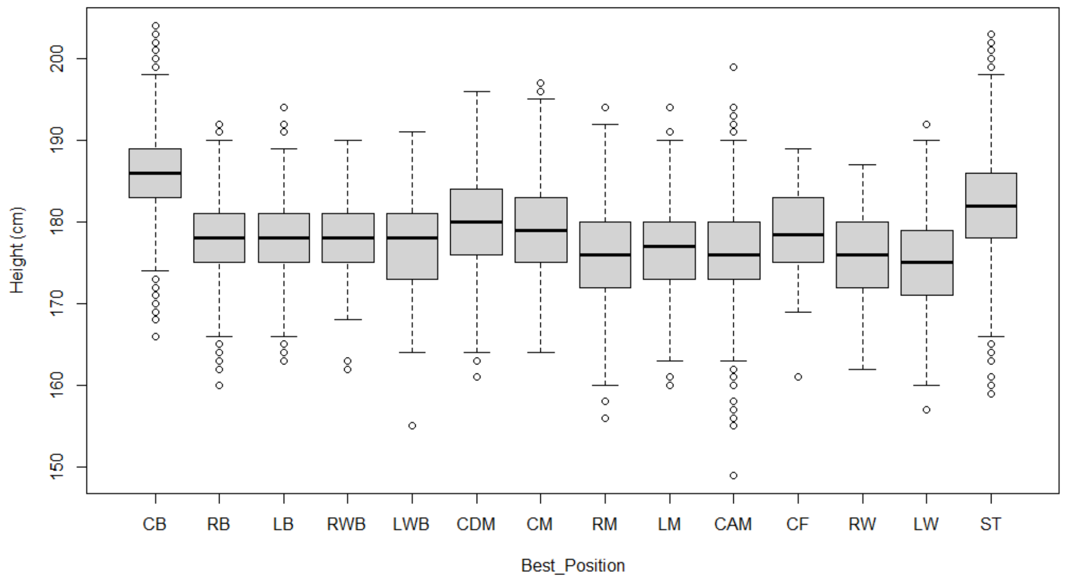


Figure 3.3: Boxplot of the response variable versus Height_cm.

For what concerns all the other quantitative covariates, plots suggest the presence of a relation between the response variable and them. More in detail:

1. with *Acceleration*, the central players' position provides lower values than the lateral ones, and the more advanced the position on the field the higher the Acceleration value. There are lots of outliers for the levels of the response variable which correspond to the higher frequencies and the same phenomenon is verified for CDM and CM positions. Moreover, ST position has the highest variance, resulting not symmetric with the other boxplots;

2. with *Aggression*, the median values are higher for defensive roles and lower for attacking ones. With the exception of CAM role, there are too few outliers;

3. with *Agility*, the lateral positions have higher values for medians. Moreover, it has been noticed high variance for CB, CDM and ST positions, with lots of outliers for the last cited position;

4. with *Balance*, the more advanced the position on the field the higher the Balance value, except for CB, CDM and ST positions which have the lowest values. The variance is different for every position, resulting in asymmetric boxplots;

5. with *Ball_ Control* in Figure 3.4, there are little groups of boxplots that are symmetric; it happens for laterals positions where staying at the left or at the right on the field does not matter. This value tends to be under 90, so only few players have great ball control capacities and for this reason the variance tends to be different for every position;

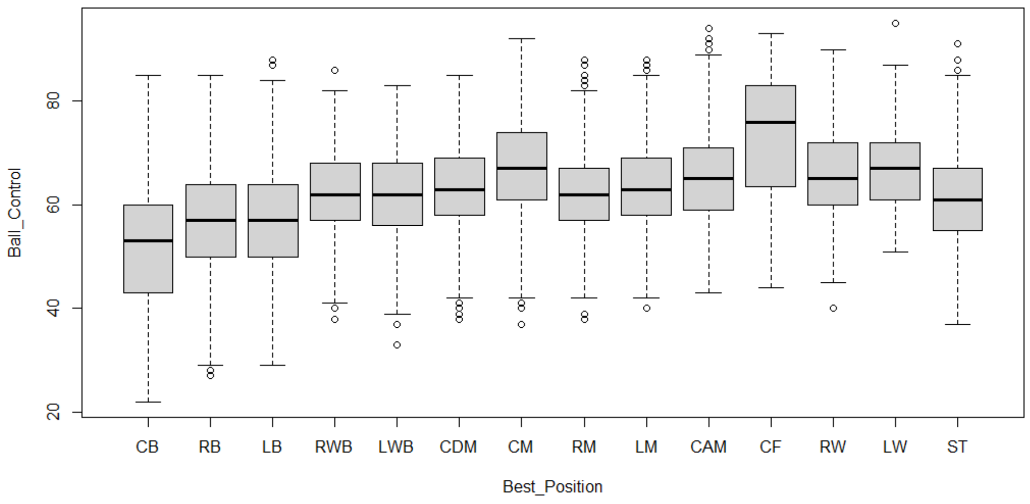


Figure 3.4: Boxplot of the response variable versus Ball_Control.

6. with *Composure*, the median values get around between 50 and 60, with CF players excels particularly around a 70 median value. The variance is large for every position;

7. with *Crossing*, the higher mean values belong to lateral position, even though a CM player which plays in the central part of the pitch has the maximum value. The variance is higher for central positions;

8. with *Curve*, the higher mean values belongs to lateral position, even though a CM player which plays in the central part of the pitch has the maximum value. The variance is higher for central positions;

9. with *Defensive_Awareness*, the median value is higher for defensive roles than attacking ones, as expected. Moreover, the variance is high for offensive midfielder positions and ST position has a lot of outliers probably due to strikers who adopt pressing on the ball carrier when the action starts from the back;
10. with *Dribbling*, the more advanced the position on the pitch the higher the median value. Anyway, this value increasing is really slight. The only extreme values which deviate from the average refer to CB and ST, respectively with a very low value and with a higher value. Moreover, CB position has a large variance;
11. with *Finishing* in Figure 3.5, the higher median values belong to offensive midfielder position and forward ones. Moreover, CB and ST positions have lots of outliers;

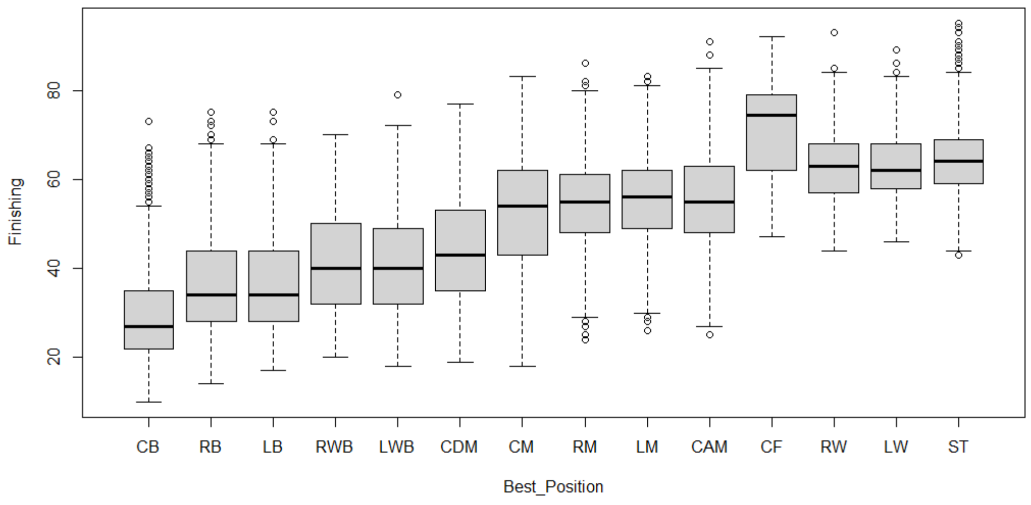


Figure 3.5: Boxplot of the response variable versus Finishing.

12. with *Free_Kick_Accuracy*, looking at the median values there are no particular trends. What it is important to report is the large variance for many positions and the large amount of outliers for defensive positions;
13. with *Heading_Accuracy*, the median value is quite constant for all the positions, except for CB, CF and ST roles which have the highest values. In addition to it, even the variance is quite constant between all the boxplots;
14. with *Interceptions*, it is clear the higher values, looking at the median values, belong to defensive positions. Anyway, variance is too large for offensive midfielder positions and ST position has a lot of outliers probably due to strikers who adopt pressing on the ball carrier when the action starts from the back;

15. with *Jumping*, the median value is quite constant for many positions, except for CB, CF and ST roles which have the highest value and except for RM, LM, CAM, RW and LW roles which have the lowest value. The variance is quite large for many offensive midfielder positions and CB and ST roles have a lot of outliers under the lower whisker;

16. with *Long_Passing*, the highest median value belongs to CM, followed by the central midfielder positions as CDM, CAM and the forward position CF. All the other median values are quite similar, except for ST which is the most advanced player on the pitch and this capacity is secondary. Anyway, the variance is very large for defensive roles, so it means there are a lot of players with great and poor capacities on making long passing. There are many outliers for CAM position;

17. with *Long_Shots*, the more advanced the position on the pitch the higher the median value, except for CM and CF positions in which they have the highest values. Anyway, there are very few players who own good capacities in long shots, it means with a value higher than 80. The variance is quite constant for the majority of the boxplots and CB position has lots of outliers above the upper whisker;

18. with *Penalties*, the more advanced the position on the pitch the higher the median value, except for CF position which have the highest value. CDM and CAM have large variance and, in addition to it, the defensive roles together with ST values have lots of outliers above the upper whisker;

19. with *Positioning*, the median values are quite similar and with a slight increment of the value for the more advanced positions on the field. This is true except for CB and CF positions which respectively own the lowest and the highest median values. The variance is quite constant for every boxplot;

20. with *Reactions*, the median values are quite similar, except for CF position which have the highest median value. The variance is quite large for every boxplot and CAM and ST positions have many outliers above the upper whisker;

21. with *Short_Passing* in Figure 3.6, the median values are quite constant between every position, but there are some trends. For instance, defensive wings have higher median values than the defensive central roles, whereas the central midfielder positions have higher median values than lateral midfielder ones. Moreover, the variance is large for the majority of the boxplots;

22. with *Shot_Power*, the more advanced the position on the pitch the higher the median value, except for CF which have the highest value. The variance is higher for the defensive and midfielder positions;

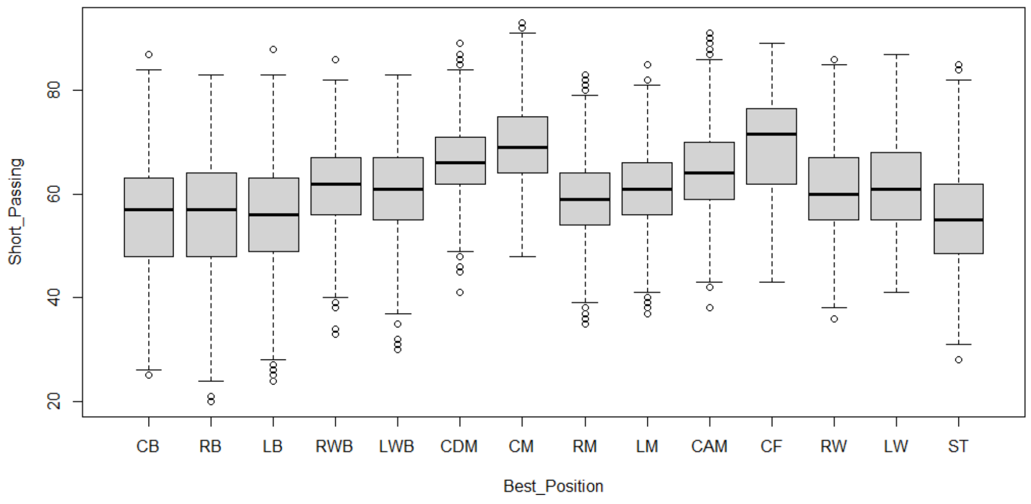


Figure 3.6: Boxplot of the response variable versus Short_Passing.

23. with *Sliding_Tackle*, the more backward positions have a constant and higher median value than the remaining positions. Moreover, the variance is quite low except for RM, LM and CAM positions. Then, ST position has lots of outliers above the upper whisker;

24. with *Sprint_Speed*, the median values are quite constant and around 70, except for CB, CDM, CM, CAM and ST positions which have lower values that get around 60. For the latter position it can be noticed the high number of outliers, in particular below the lower whiskers of the boxplots;

25. with *Stamina*, all the median values get around 60 and 70, so a limited range of values where the first and third quantiles are quite near to the median values. Anyway, there are slight differences in variance and some outliers in CB and ST positions, so this attribute could be considered as significant;

26. with *Standing_Tackle* in Figure 3.7, the more backward positions have a constant and higher median value than the remaining positions. Moreover, the variance is quite low except for RM, LM and CAM positions. Then, ST position has lots of outliers above the upper whisker;

27. with *Strength*, looking at the median values there are no particular trends, except for CB and ST positions which have the highest median value;

28. with *Vision*, the more advanced the position on the pitch the higher the median value, except for CM and CF positions which have the highest values. It is important to

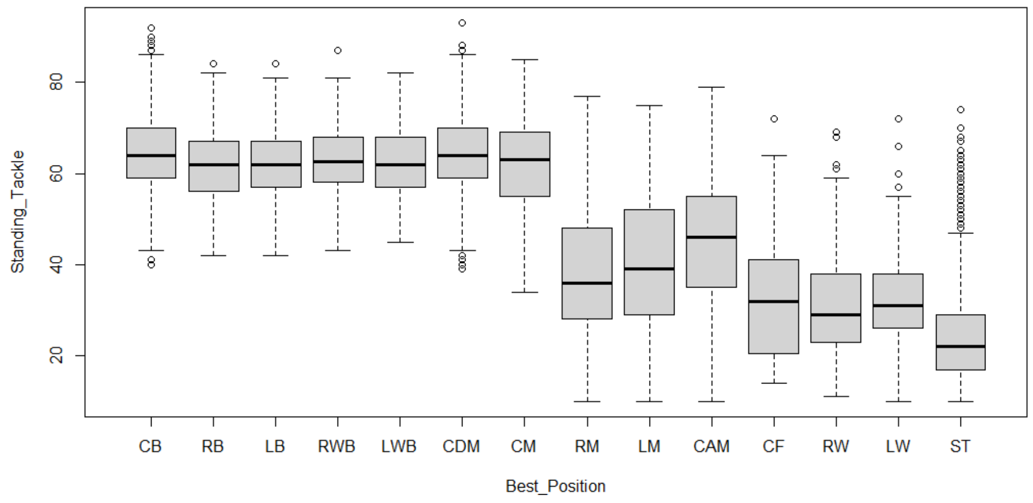


Figure 3.7: Boxplot of the response variable versus Standing_Tackle.

notice CB, RM, CAM and ST positions have some outliers above the upper whiskers of the boxplots;

29. with *Volleys*, the more advanced the position on the field the higher the Volleys value. The only position which does not respect this statement is CF one, where the median value get around on 70. The central defensive roles have lots of outliers above the upper whisker probably due to acrobat defensive players with great capabilities when hit the ball in the air.

3.2.4 Possible interactions between covariates

An additional graphical investigation considers interactions between quantitative variables and the response variable. This type of graphical analysis has been conducted using *ggplot*, which allows to create scatterplots for analyzing the relationship between three or more variables. The visualization method is the so-called "grouping", where x and y axes refer to numerical variables and every point in the graph is coloured referring to the levels of a categorical variable (Kabacoff, 2018). To simplify the visualization of data regarding the response variable, it has been opted to group the players' position and simplifying the number of levels to three. The assumption is the levels refer to geographical position on the field, which are *Defenders*, *Midfielders* and *Forwards*, as indicated in the Section § 2.5.

Figure 3.8 illustrates the association between *Sprint_Speed* and *Acceleration*. It can be assumed the relation is linear with a positive correlation, indeed the higher the *Sprint_Speed* value the higher the *Acceleration* value. Going deeper, the response variable levels are not well-separated, suggesting there could be an interaction between the two quantitative variables.

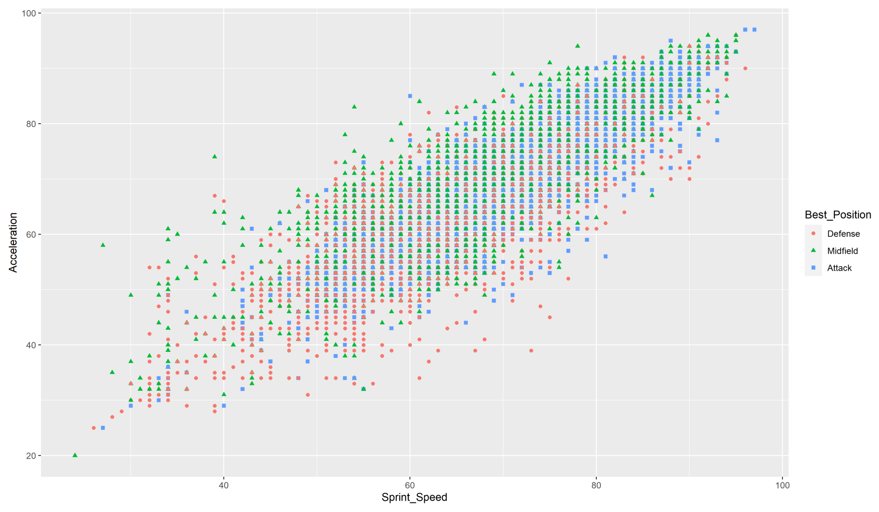


Figure 3.8: Plot of Sprint_Speed versus Acceleration.

Figure 3.9 illustrates the association between *Ball_ Control* and *Dribbling*. It can be assumed the relation is more complex than linear with a positive correlation, indeed the higher the *Ball_ Control* value the higher the *Dribbling* value. Going deeper, the group of defenders are quite well-separated and they are spread at below left part of the plot, whereas the midfielders and forwards groups are randomly distributed in the plot at above right part of the plot with higher values because more skilled. So, it can be supposed there could be an interaction between the two quantitative variables.

Figure 3.10 illustrates the association between *Free_Kick_Accuracy* and *Curve*. It can be assumed the relation is linear with a positive correlation and a large variance, indeed the higher the *Free_Kick_Accuracy* value the higher the *Curve* value. Going deeper, there is no defined separation of the response variable levels, even if the majority of midfielders and forwards own the higher values. It suggests there could be an interaction between the two quantitative variables.

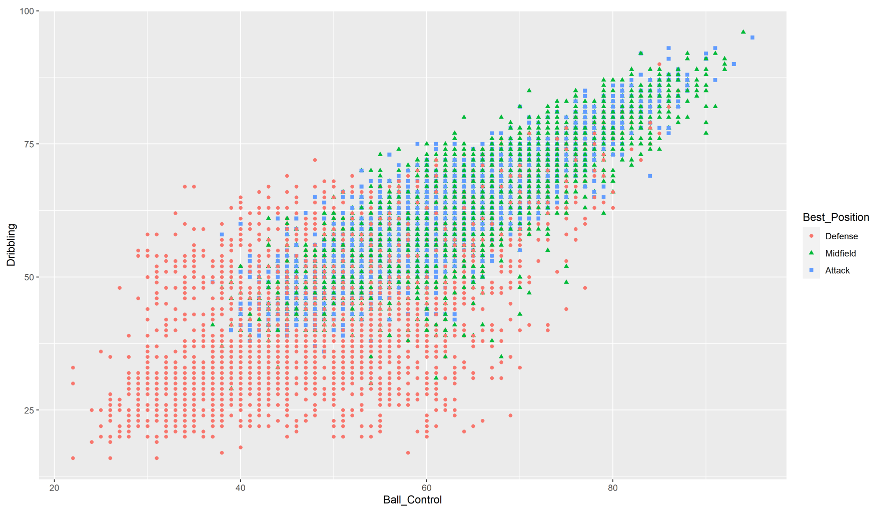


Figure 3.9: Plot of Ball_Control versus Dribbling.

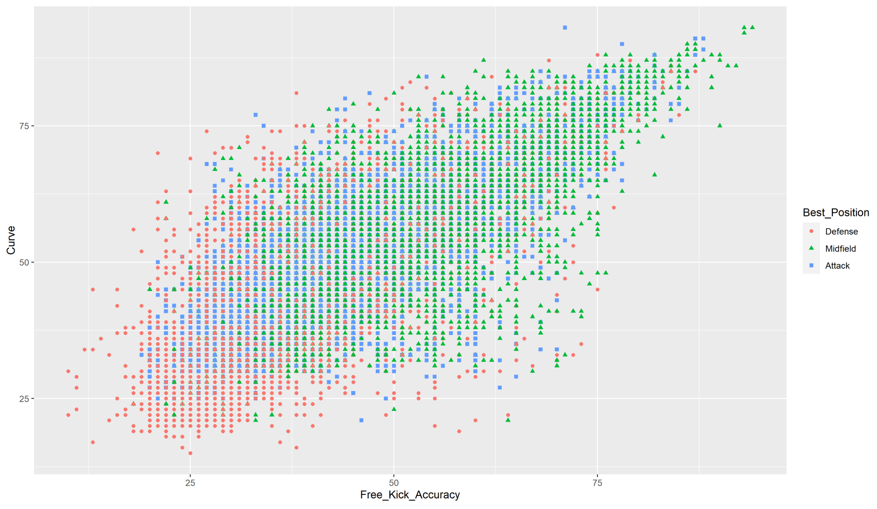


Figure 3.10: Plot of Free_Kick_Accuracy versus Curve.

Figure 3.11 illustrates the association between *Sliding_Tackle* and *Standing_Tackle*. It is very clear the relation is linear with a strong positive correlation, indeed the higher the *Sliding_Tackle* value the higher the *Standing_Tackle* value. Going deeper, the highest values belongs to defenders and the response variable levels are quite well-separated, suggesting there could not be an interaction between the two quantitative variables.

Figure 3.12 illustrates the association between *Vision* and *Long_Passing*. It can be assumed the relation is more complex than linear with a positive correlation and a large variance, in-

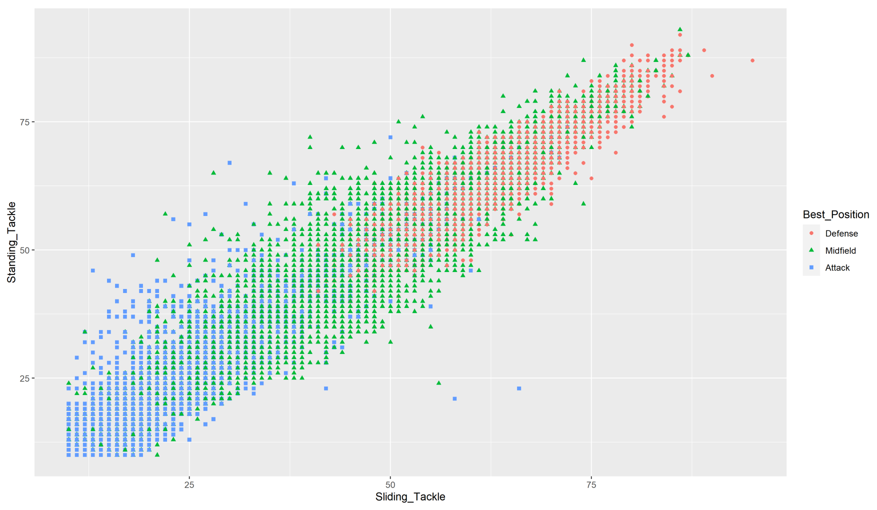


Figure 3.11: Plot of Sliding_Tackle versus Standing_Tackle.

deed the higher the *Vision* value the higher the *Long_Passing* value. The response variable levels are quite well-separated, suggesting there could not be an interaction between the two quantitative variables.

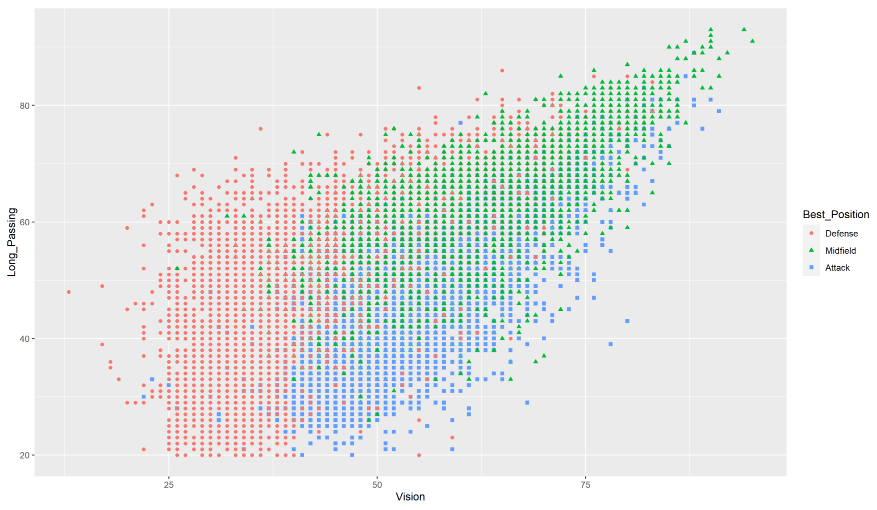


Figure 3.12: Plot of Vision versus Long_Passing.

Figure 3.13 illustrates the associations between *Agility* and the covariates *Dribbling* and *Ball_Control*, where they have a quite similar behaviour. For both the scenarios, it can be assumed the relation is linear with a positive correlation and a large variance, indeed the higher the *Dribbling* and *Ball_Control* values the higher the *Agility* value. Defenders are well-separated than the other two *Best_Position* levels which are mixed altogether, suggesting there could not be an interaction between the two quantitative variables for both the scenarios.

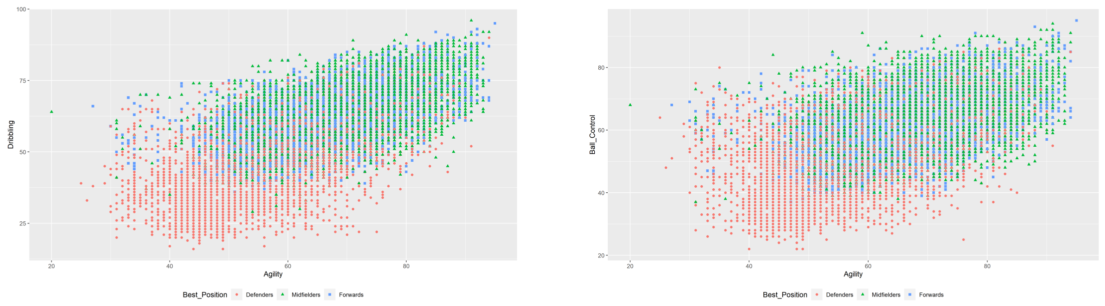


Figure 3.13: Plot of Agility versus Dribbling and Ball_Control.

3.3 Correlation matrix

Finally, it is useful to analyze the correlation matrix, in which only quantitative variables are involved. This type of graphical analysis has been conducted using *ggcorrplot R* package. This final step is necessary to guarantee the absence of multicollinearity, a phenomenon according to which two or more highly linear correlated quantitative variables become not significant for a predictive model. If two variables are highly correlated and they hide the real effect of another significant variables, one of them needs to be deleted. The removal process allows to reduce the number of quantitative covariates, removing the ones which are very similar with the other maintained in the dataset.

Figure 3.14 shows the correlation matrix, where in every tile is displayed the correlation value between the corresponding x-value and y-value. This value is represented both with a color and with the exact coefficients of Pearson's correlation. The redder the tiles the higher and positive the correlation is, viceversa the more blue the tiles the higher and negative the correlation is. In the latter both cases the relationship is stronger, whereas white tiles means no correlation and so independence between variables. More in detail, there are some tiles without the exact correlation value but only an empty white tile; it means the correlation coefficient of the involved variables is not significant due to a p-value higher than 0.05, which is the default significance level.

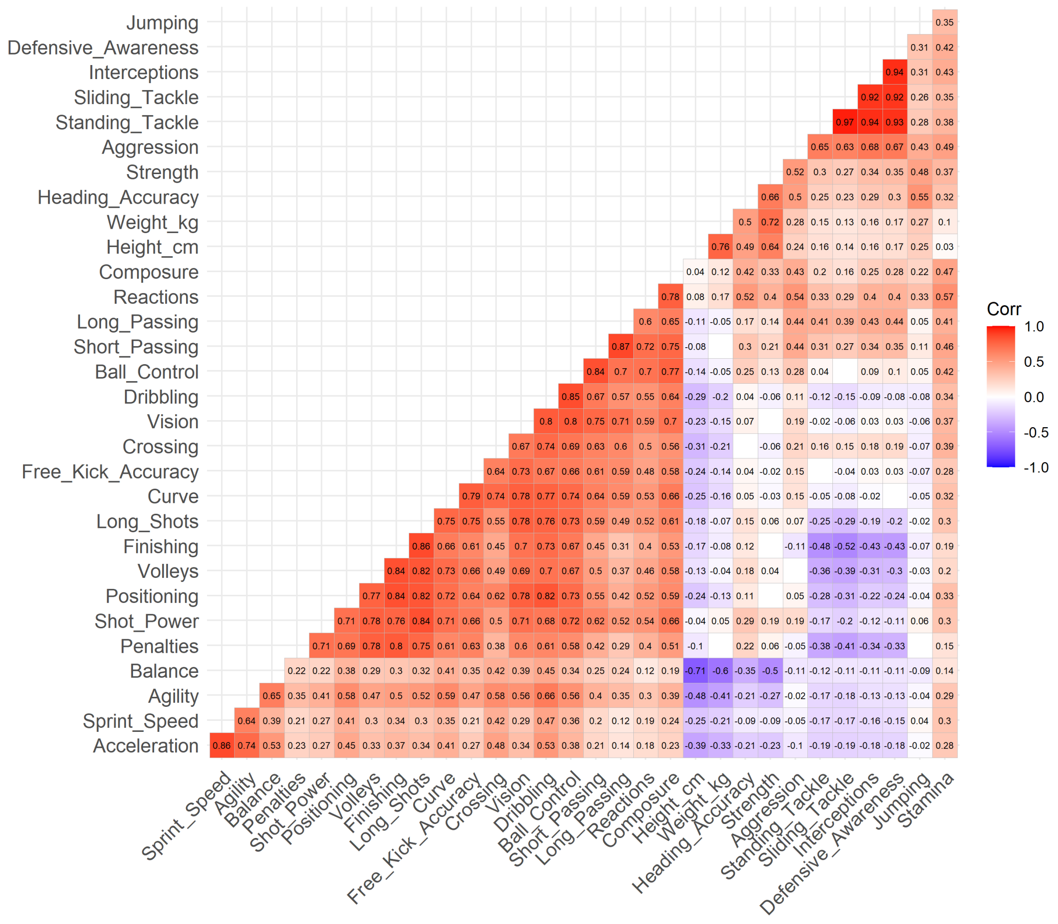


Figure 3.14: Correlation matrix with associated p-values.

As expected, there are plenty of quantitative variables which are highly correlated, the mostly with a positive correlation coefficients. The more related variables are listed below.

- *Standing_Tackle* and *Sliding_Tackle* have a positive high correlation coefficient of 0.97. These two variables are highly correlated because they are two types of different tackle with the same aim to take away the ball to the opponent.

- *Interceptions* and *Defensive_Awareness* have a positive high correlation coefficient of 0.94. These two variables are highly correlated because a player with high marking ability has a higher probability to intercept balls.

- *Standing_Tackle* and *Interceptions* have a positive high correlation coefficient of 0.94. These two variables are highly correlated because interceptions can be made tackling the opponent player while standing.

- *Standing_ Tackle* and *Defensive_Awareness* have a positive high correlation coefficient of 0.93. These two variables are highly correlated because a standing tackle is a defensive ability.
- *Sliding_ Tackle* and *Defensive_Awareness* have a positive high correlation coefficient of 0.92. These two variables are highly correlated because a sliding tackle is a defensive ability.
- *Sliding_ Tackle* and *Interceptions* have a positive high correlation coefficient of 0.92. These two variables are highly correlated because interceptions can be made tackling the opponent player using legs.
- *Short_Passing* and *Long_Passing* have a positive high correlation coefficient of 0.87. These two variables are highly correlated because they are two types of passing with the same aim to give the ball to a teammate.
- *Acceleration* and *Sprint_Speed* have a positive high correlation coefficient of 0.86. These two variables are highly correlated because they are both related to the running speed.
- *Finishing* and *Long_Shots* have a positive high correlation coefficient of 0.86. These two variables are highly correlated because they are two types of shots, which depend on the distance from which they are carried out, with the same aim to score a goal.
- *Dribbling* and *Ball_ Control* have a positive high correlation coefficient of 0.85. These two variables are highly correlated because they refer to the management of the ball.
- *Balance* and *Weight_kg* have a negative high correlation coefficient of -0.6. These two variables are highly correlated because in most cases the heavier the player, the less balanced the player.
- *Balance* and *Height_cm* have a negative high correlation coefficient of -0.71. These two variables are highly correlated because in most cases the higher the player, the less balanced the player.

To avoid or mitigate multicollinearity phenomenon, some variables needs to be removed. For this task, only the covariates pairs with a correlation coefficient higher than 0.9 are considered. The covariates involved in this deletion process are *Standing_ Tackle*, *Sliding_ Tackle*, *Interceptions* and *Defensive_Awareness* and they describe specific defensive roles. The pairs, written in *x-y* notation, are: *Standing_ Tackle-Sliding_ Tackle*, *Interceptions-Defensive_Awareness*, *Standing_ Tackle-Interceptions*, *Standing_ Tackle-Defensive_Awareness*, *Sliding_ Tackle-Defensive_Awareness* and *Sliding_ Tackle-Interceptions*. It is clear that every single covariate is highly related to each one listed above. So, we can maintain only one covariate and the choice

falls in *Interceptions* because it is involved in the lowest correlation coefficient among all the ones considered and it encloses the defensive roles of a player.
The updated correlation matrix is available in Figure 3.15. At the end of this process, three variables have been removed and the dataset contains 17018 rows and 34 columns.

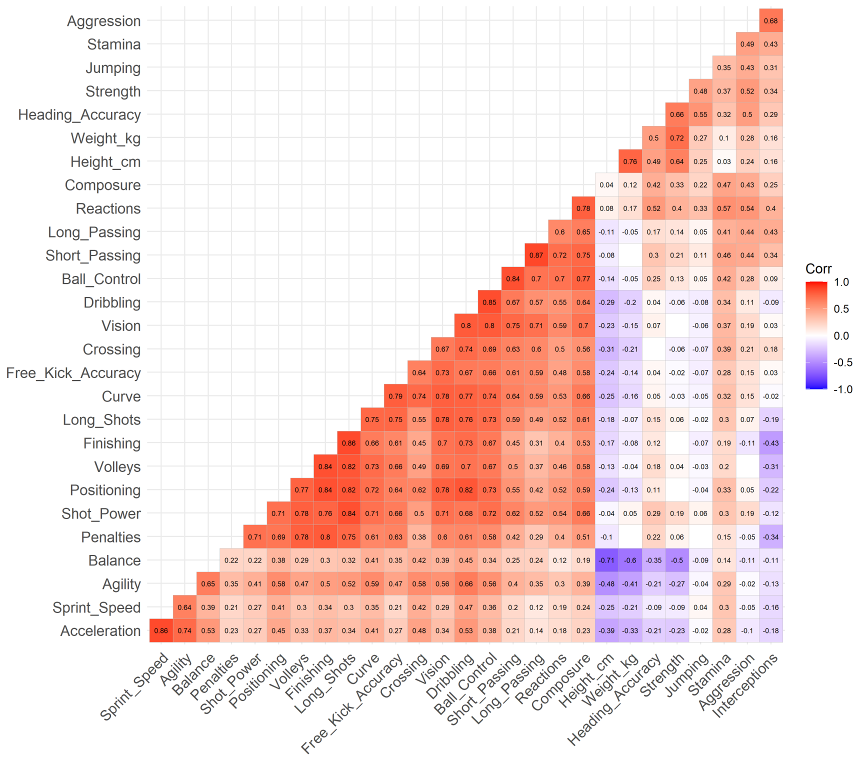


Figure 3.15: Updated correlation matrix after removal process.

4 Data Mining Techniques

This chapter focuses on describing Data Mining techniques which belong to the classification domain. The aim of Data Mining methods is to explore and evaluate the most significant relationships between the categorical response variable *Best_Position* and the covariates. The techniques involved in this chapter are Multinomial Logistic Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Ridge Regression and Lasso. Further on, the resulting models will be used for making prediction and a comparison between Data Mining techniques and other Machine Learning techniques deepened in Chapter § 6 will be investigated.

4.1 Premises

4.1.1 Multinomial distribution

The Multinomial distribution (Agresti, 2013, Chapter 1) is a specific distribution for categorical data in which the response variable has more than two possible outcomes. Suppose there are N independent and identical observations which can assume $k = 1, \dots, K$ possible discrete values. Let $y_{ik} = 1$ if the observation $i = 1, \dots, N$ belongs to the class k and $y_{ik} = 0$ otherwise. Let $y_i = (y_{i1}, \dots, y_{iK})^T$ the vector of responses for observation i, with $\sum_{k=1}^{K} y_{ik} = 1$. The response y_{iK} can be considered as redundant due to linear dependency on all other responses. Let $n_k = \sum_{i=1}^{N} y_{ik}$ represents the number of observations falling into class *k*. At this point, $(n_1, ..., n_K)^T$ is the vector of counts for each class. Let $\pi_k = Pr(Y_{ik} = 1)$ represent the probability of falling into class *k* for each observation. Since $\sum_{k=1}^{K} n_k = n$ and $n_K = n - (n_1 + ... + n_{K-1})$, the multinomial probability function is *K-1*-dimensional with expression:

$$Pr(n_1, ..., n_{K-1}) = \binom{n}{n_1, ..., n_K} \pi_1^{n_1} \cdots \pi_K^{n_K}.$$

The mean, the variance and the covariance associated to the multinomial distribution are

$$E(n_k) = n\pi_k \qquad VAR(n_k) = n\pi_k(1 - \pi_k) \qquad COV(n_a, n_b) = -n\pi_a\pi_b,$$

respectively. Finally, the marginal distribution of each count n_k is binomial.

4.2 Multinomial Logistic Regression

The Multinomial Logistic Regression (James et al., 2021, Chapter 4) is a supervised learning technique for predicting a qualitative response variable. It is an extension of the Logistic Regression technique, with the difference that Multinomial Logistic Regression allows the response variable to contain more than two discrete classes. Before starting the modeling process, it is necessary to fix a relevant base category K of the response variable called *baseline*, which the interpretation of the model coefficients will be based on.
Multinomial Logistic Regression allows to estimate the probability for each class as:

$$\pi_k(x_i) = Pr(Y = k|X = x_i) = \frac{e^{\beta_{k_0} + \beta_{k_1} x_{i_1} + ... + \beta_{k_p} x_{i_p}}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l_0} + \beta_{l_1} x_{i_1} + ... + \beta_{l_p} x_{i_p}}}$$

for the k-th class of the response variable where $k = 1, ..., K - 1$ and

$$\pi_K(x_i) = Pr(Y = K|X = x_i) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l_0} + \beta_{l_1} x_{i_1} + ... + \beta_{l_p} x_{i_p}}}$$

for the $K - th$ baseline level of the response variable, where:

- $X = (X_1, ..., X_N)^T$ is the input vector composed by $i = 1, ...N$ observations, in which every X_i contains $p = 1, ..., P$ covariates;
- β_{k_0} is the intercept of the k-th response variable level;

- $\beta_{k_1}, ..., \beta_{k_p}$ are the slope coefficients of the k-th response variable level.

Generally, the most common relevant quantity used for describing a relation between a specific level of response variable given the model covariates and the baseline is the so-called *log odds*, or simply *logit*. It allows to relate a transformation of the response variable to the covariates using a linear model (Pace and Salvan, 1997, Chapter 6). A log odds is defined as a function of covariates as follows:

$$\ln\left(\frac{\pi_k}{\pi_K}\right) = \ln\left(\frac{Pr(Y = k|X = x_i)}{Pr(Y = K|X = x_i)}\right) = \beta_{k_0} + \beta_{k_1}x_{i_1} + ... + \beta_{k_p}x_{i_p}, \tag{4.1}$$

which allows to interpret how a variation of one unit in each coefficient changes the log-odds of going from level K to level $k = 1, ..., K-1$. Similarly, *relative risk ratio* or simply *odds* is a quantity obtained exponentiating both sides in formula (4.1), which allows to simplify the interpretation of the coefficients as a variation of one unit change in each coefficient changes the risk of falling in level $k = 1, ..., K - 1$ compared to the baseline K.
There are some assumptions which would be respected to obtain reliable multinomial logistic regression models.

- The *independence of irrelevant alternatives* (IIA) assumption supposes that the relative likelihood on falling in a $k = 1, ..., K-1$ level compared to the baseline K is not influenced by the addition of other levels into the response variable.
- The outcome must be categorical and the response variable has more than two categories.
- The log odds of the outcome have a linear relationship with any covariate.
- Errors are independent.
- Collinearity should be avoided, so the phenomenon in which there is high correlation between one covariate to another one.

The parameter vector $\beta_k = (\beta_{k_0}, \beta_{k_1}, ..., \beta_{k_p})^T$ is typically estimated using maximum likelihood (Agresti, 2013, Chapter 8). Let $y_i = (y_{i1}, ..., y_{iK})^T$ represents the multinomial trial for subject *i*, where $y_{ik} = 1$ identifies a response in the level j and $yik = 0$ vice versa. Then, $\sum_{k=1}^{K} y_{ik} = 1$. Let $x_i = (x_{i1}, ..., x_{ip})^T$ represents the covariates values for subject *i*. Since

$\pi_J = 1 - (\pi_1 + ... + \pi_{J-1})$ and $y_{iJ} = 1 - (y_{i1} + ... + y_{i,J-1})$, the subject *i* contributes to the logarithm of the likelikood

$$\begin{aligned}\prod_{k=1}^{K} \pi_k(x_i)^{y_{ik}} &= \sum_{k=1}^{K-1} y_{ik} \ln \pi_k(x_i) + \left(1 - \sum_{k=1}^{K-1} y_{ik}\right) \ln \left[1 - \sum_{k=1}^{K-1} \pi_k(x_i)\right] \\ &= \sum_{k=1}^{K-1} y_{ik} \ln \frac{\pi_k(x_i)}{1 - \sum_{k=1}^{K-1} \pi_k(x_i)} + \ln \left[1 - \sum_{k=1}^{K-1} \pi_k(x_i)\right].\end{aligned}$$

Now, assuming N independent observations, the log likelihood becomes:

$$\begin{aligned}\prod_{i=1}^{N} \left[\prod_{k=1}^{K} \pi_k(x_i)^{y_{ik}}\right] &= \sum_{i=1}^{N} \left\{\sum_{k=1}^{K-1} y_{ik}(\beta_k^T x_i) - \ln \left[1 + \sum_{k=1}^{K-1} e^{\beta_k^T x_i}\right]\right\} \\ &= \sum_{k=1}^{K-1} \left[\beta_{k_0} \left(\sum_{i=1}^{N} y_{ik}\right) + \sum_{p=1}^{P} \beta_{kp} \left(\sum_{i=1}^{N} x_{ip} y_{ik}\right)\right] \\ &= \sum_{i=1}^{N} \ln \left[1 + \sum_{k=1}^{K-1} e^{\beta_k^T x_i}\right]. \qquad (4.2)\end{aligned}$$

The log likelihood function is concave and the Netwon-Raphson algorithm provides the maximum likelihood estimates.

4.3 Automatic model selection

Finding the best ever multinomial logistic regression model is an strenuous activity when the number of covariates is large due to the huge number of possible models to compare. Anyway, automatic stepwise selection techniques are available to look for and choose the best model according to specific metrics. There are three types of stepwise selection.

- **Forward selection** starts with the only intercept 1 and at each step it adds one covariate which improves more the previous model. This process goes one until reaching the full model, that contains all the covariates, or when there is no chance on improving the current model.

- **Backward selection** starts with a model including all the covariates and at each step it deletes one nonsignificant covariate at a time. This process goes on until there are only significant covariates in the resulting model or the null model, that contains only the intercept 1, is reached.

- **Hybrid selection** is a mixture between forward selection and backward selection. Indeed, the initial model is composed by the only intercept 1 and at each step it adds one covariate which improves more the previous model. If a variable included in the model does not improve it, then the variable is removed. This process goes on until there is no chance on improving the current model.

In the classification domain, variable selection can be performed using the following metrics:

- *AIC*[G] (Akaike Information Criterion) (Azzalini and Scarpa, 2012, Chapter 3)

  $$AIC = Deviance + 2p, \tag{4.3}$$

  where *Deviance* is the log-likelihood of the fitted model multiplied by 2 and it is influenced by the $2p$ penalization term in which *p* is the number of estimated parameters. AIC gives a score based on the fitness of the model and its complexity, penalizing less complex models. The smaller the AIC value, the better fit of the model and the lower its test error.

- *BIC*[G] (Bayesian Information Criterion) (Hastie, Tibshirani, and Friedman, 2009, Chapter 7)

  $$BIC = Deviance + pln(n), \tag{4.4}$$

  where *Deviance* and p are defined as above and it is influenced by the $pln(n)$ penalization term in which *n* is the number of observations in the training set. BIC gives a score based on the fitness of the model and its complexity. Differently from AIC, it tends to penalize more complex models when $ln(n) > 2$. The smaller the BIC value, the better fit of the model and the lower its test error.

The advantage of both the listed metrics is that they do not need models to be nested. In addition to it, they are based on asymptotic arguments, above all when the number of observations is high. Automatic model selection algorithms are efficient only when the number of observation is higher than the number of covariates. Moreover, they are useful for obtaining quickly information about the relationships between the response variable and the significant covariates. On the other hand, these algorithms does not take into account the variability associated to the model choice.

4.4 Linear Discriminant Analysis

The *LDA*[G] (Linear Discriminant Analysis) (James et al., 2021, Chapter 4) is a modeling approach for estimating the posterior probability that an observation belongs to the k-th class given the predictors values of the observation as follows:

$$p_k(x) = P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{j=1}^{K} \pi_j f_j(x)}, \tag{4.5}$$

where π_k is the prior probability that a randomly chosen observation belongs to the k-th class, and $f_k(x)$ is the density function of $X = x$ for an observation that comes from the k-th class. Assumed that the number of predictors *p* is greater than one, this technique is similar in form to (multinomial) logistic regression. It is assumed that $X = (X_1, ..., X_p)$ follows a multivariate normal distribution within each level of the response variable, with a class-specific mean vector and a common covariance matrix where all the variances $\sigma_1^2 = ... = \sigma_K^2$ are equal due to the homogeneity assumption. Moreover, it allows to discriminate a response variable divided into more than two levels. Differently from (multinomial) logistic regression, LDA does not compute directly these estimates. Indeed, first it models the distribution of X for each response variable level, then Bayes' theorem is used to obtain $P(Y|X)$. The unknown quantities which needs to be estimated in formula (4.5) are π_k and $f_k(x)$. In particular,

$$\pi_k = \frac{n_k}{n}$$

where n_k is the number of observations in the training set for k-th class and n is the total number of observation in the training set; moreover,

$$f_k(x) = \frac{1}{(2\pi_k)^{p/2}|\Sigma|^{1/2}} e^{\frac{1}{2}(x-\mu_k)^T|\Sigma|^{-1}(x-\mu_k)}$$

where $f_k(x)$ is assumed as multivariate normal and μ_k and Σ are respectively the mean vector with p elements and the *p*×*p* covariance matrix of X that is common for each class. Substituting π_k and $f_k(x)$ terms in formula (4.5) and applying log transformation to it, an observation $X = x$ is assigned to the class for which

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \tag{4.6}$$

is largest. Formula (4.6) defines the decision boundaries and is called *discriminant function*. It is a linear combination of the covariates and it is built K-1 times for a response variable with K levels.

4.5 Regularization methods

The management of data can be tricky when they are high-dimensional, in other words when the number of the predictors *p* is equal or larger than the number of subjects *n*. In this scenario, a multinomial logistic regression model could not offer satisfactory metrics values, feeding the problems of identifiability and efficiency of the model. More in detail, the estimates obtained by the maximum likelihood function could be difficult to calculate, or even impossible, or they could have large standard errors. Moreover, a complex model could incur in overfitting, which means that a model captures all the (even noised) details by training data and not the general trend with the consequence to obtain unsatisfactory results in prediction with unseen data. Regularization methods have been designed to face these problems. The basic idea of regularization methods is a shrinkage of the coefficients estimates towards zero in order to reduce the variability of the estimates. Regularization is possible applying a penalizing function, named *shrinkage penalty*, to the likelihood function. Anyway, it is necessary to pay attention to the trade-off between the likelihood function and the shrinkage penalty. Indeed, higher penalty means more penalization in less realistic values of the parameters estimates, but in the other hand it means less information. Another detail is that the intercepts of a multinomial logistic regression model are not involved in the penalization process. Finally, the values of the variables need a standardization process in order to avoid problems with scale effects on results. In the thesis we will focus on two famous regularization methods, namely, Ridge Regression and *Lasso*[G], that are briefly described below.

4.5.1 Ridge Regression

Ridge Regression (James et al., 2021, Chapter 6) is a regularization method with the target of shrinking the coefficients estimates close to zero, including all the predictors in the final model. With this method, the coefficients are estimated by minimizing the log likelihood plus the L2 regularization shrinkage penalty as follows:

$$\beta_{ridge} = \arg\min_{\beta} \left\{ l(\beta|\theta) + \lambda \sum_{j=1}^{P} \beta_j^2 \right\},$$

where $l(\beta|\theta)$ is the log-likelihood found in formula (4.2), and $\lambda \sum_{j=1}^{P} \beta_j^2$ represents the shrinkage penalty which in literature is called *weight decay*. More in detail, λ is the tuning pa-

rameter of the penalization term which assumes natural values and it is calculated separately. In the case of $\lambda = 0$, the shrinkage penalty has no effect, obtaining a model with the same coefficients estimates found with the multinomial logistic regression model. On the other hand, when $\lambda > 0$ the effects of the shrinkage penalty increase as λ increases, reducing the variance and the model complexity. It is necessary to pay attention which λ value to assign. Indeed, the larger the λ value, the more the coefficients estimates are close to zero, with the consequence that the association between the response variable and the predictors is drastically reduced. Moreover, the larger the lambda value, the less flexible the model with the consequence that the variance decreases but the bias increases. So, the choice of λ plays an important role on the trade-off management between the likelihood and the penalization term. Indeed, the set of coefficients estimates changes every time λ value changes. For this reason, it is necessary to apply cross validation to find the best possible λ value (Hastie, Tibshirani, and Friedman, 2009, Chapter 3).
Finally, the Ridge Regression method can be viewed as an optimization problem due to a size constraint *s* applied on the sum of squared coefficients estimates. The problem can be formulated as follows:

$$\beta_{ridge} = \arg\min_{\beta} \left(l(\beta|\theta)\right)^2$$
$$s.t. \sum_{j=1}^{P} \beta_j^2 \leq s.$$

4.5.2 Lasso

Lasso, acronym of *Least Absolute Shrinkage and Selection Operator* (James et al., 2021, Chapter 6), is an alternative to Ridge Regression as a regularization method. The difference is that Lasso performs both the shrinking of the coefficients estimates close to zero and variable selection. With this method, the coefficients are estimated by minimizing the log likelihood plus the L1 regularization shrinkage penalty as follows:

$$\beta_{lasso} = \arg\min_{\beta} \left\{ l(\beta|\theta) + \lambda \sum_{j=1}^{P} |\beta_j| \right\},$$

where $l(\beta|\theta)$ is the log-likelihood found in formula (4.2), and $\lambda \sum_{j=1}^{P} \beta_j^2$ represents the shrinkage penalty. The functioning of λ value is the same as in Ridge Regression, with the

difference that due to the L1 regularization shrinkage penalty some coefficients estimates are forced to be exactly zero when λ is sufficiently large. This behaviour makes a Lasso model easier to interpret because only a subset of the covariates is involved. Like in Ridge Regression, cross validation is necessary to find the best possible λ value for obtaining a significant model. Depending on the chosen λ, the main difference is that Lasso can contain whatever number of covariates while Ridge Regression includes always all the covariates (Hastie, Tibshirani, and Friedman, 2009, Chapter 3).
Finally, the Ridge Regression method can be viewed as an optimization problem due to a size constraint *s* applied on the sum of absolute coefficients estimates. The problem can be formulated as follows:

$$\beta_{lasso} = \arg\min_{\beta} \left(l(\beta|\theta)\right)^2$$
$$s.t. \sum_{j=1}^{P} |\beta_j| \leq s.$$

5
Data Mining Results

This chapter focuses on the implementation and the discussion of the prediction results obtained from the Data Mining techniques described in Chapter § 4. The details about the implementation will be supported by *R* (R Core Team, 2022) code and graphical analysis.

5.1 Premises

5.1.1 Dataset split

Every technique exploits the dataset obtained after its preliminary analysis made in Chapter § 3. Before applying the implementation of each technique, the dataset needs to be split into two parts, namely training set and test set. According to the literature, a sample 80/20 from the dataset is adopted. In other words, the 80% of the dataset observations are reserved for the training set and the remaining 20% for the test set. Following this procedure, the resulting sets could be unbalanced, with high probability that a model could learn a pattern for the most frequent classes. To avoid this problem, a sample 80/20 for every player's position from the dataset has been applied. In other words, the training set contains the 80% of the dataset observations for each player's position and the remaining 20% of the dataset observations for each player's position for the test set. Such a choice allows to create two balanced sets, maintaining the same multinomial distribution seen in Section § 3.2.1. At the end, training set contains 13608 observations with 34 covariates and test set contains 3410 observations with 34 covariates. The code concerning the dataset split is shown in Listing § A.2.

5.1.2 Metrics

The metrics used to evaluate a model after the training phase are listed below.

- **Training Accuracy**, which is the ratio between the sum of the training set correct predicted values for each response variable level and the total number of training set observations. The correct predicted values can be accounted for by the confusion matrix diagonal computed using the *confusionMatrix* function of the *R caret* package (see *Function confusionMatrix*).
- **AIC**, already defined in formula 4.3. This metric is not available for LDA technique.
- **BIC**, already defined in formula 4.4. This metric is not available for LDA technique.

The metrics used for evaluating a model in prediction phase are listed below.

- **Test Accuracy**, which is the ratio between the sum of the test set correct predicted values for each response variable level and the total number of test set observations. The Test Accuracy computation is performed as in *Training Accuracy* metric.
- ***AUC***[G], acronym for *Area Under the ROC Curve* (James et al., 2021, Chapter 4), is the measure of the overall class-specific performance of a classifier. Let TP be the number of true positives, TN the number of true negatives, FP the number of false positives, and FN the number of false negatives. Then, AUC summarizes all the possible thresholds given by:
  - *Sensitivity*, called also *Recall* or *True Positive Rate* in Machine Learning field, which is formulated as $\frac{TP}{TP+FN}$ for each level $k = 1, ..., K$ and it is the percentage of the correct predicted values identified considering both TP and FP;
  - *Specificity*, called also *True Negative Rate*, which is formulated as $\frac{TN}{TN+FP}$ for each level $k = 1, ..., K$ and it is the percentage of the predicted values correctly identified not classified as the reference level considering both TN and FP.

  Sensitivity and specificity are crucial for specifying the best trade-off for correctly classifying an observation. The AUC value ranges from 0 to 1 and it derives from the *ROC*[G] (Receiver Operating Characteristics) curve. The larger the AUC, the better the model in classifying correctly. In this thesis, the ROC curve cannot be displayed because it is available only for binary classifiers. Moreover, the final AUC is called *multiclass AUC* because it is the mean of all the AUC values computed for each response variable level (Hand and Till, 2001). The computation of the AUC value is made through the *multiclass.roc* function, available from the *R pROC* package (see *Function multiclass.roc*).

Another possible metric to analyze could be *Precision*, called also *Positive Predictive Value*, which is formulated as $\frac{TP}{TP+FP}$ for each level $k = 1, ..., K$ and it is the percentage of the correct predicted values identified considering both TP and FP. Anyway, *Recall* has been preferred to *Precision* in the thesis because the false negatives are more costly than the false positives in this classification domain.

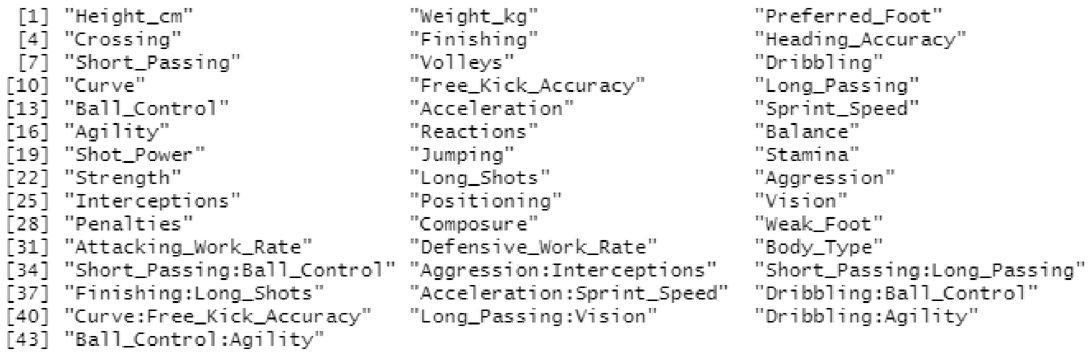
5.2 Multinomial Logistic Regression

Multinomial Logistic Regression is the first Data Mining technique we consider, applied using the *multinom* function, available from the *R nnet* package (see *Function multinom*). The *multinom* function allows to fit multinomial log-linear models with training data through a neural network with one hidden layer. The neural network is a classification network, in which the number of outputs is equal to the number of response variable levels and the class is selected through the *softmax* function. Due to the large number of variables in the model, the default number of iterations is modified to 10000, in order to facilitate convergence. The BFGS quasi-Netwon optimization method is used as optimization technique.
The modeling process starts adding all the covariates inside the model, recognizable as *starting model*. At first sight, all the covariates of the *starting model* are significant, even though lots of levels associated to the factors are not. For what concerns the metrics values, the Training Accuracy value is 0.826, the AIC value is 14448.55 and the BIC value is 19042.3. To improve the *starting model*, some interactions have been added through a trial-and-error process, in order to better explain the variability of the data. Let *updated model* be the *starting model* with the addition of the interactions suggested in Section § 3.2.4, which are *Acceleration*:*Sprint_Speed*, *Ball_Control*:*Dribbling*, *Free_Kick_Accuracy*:*Curve*, *Vision*:*Long_Passing*, *Agility*:*Dribbling* and *Agility*:*Ball_Control*. The *updated model* can be compared with the *starting model* as they are nested. The comparison can be carried out with *anova* function provided by *R stats* package (see *Function anova*), in which its aim is to check the goodness of a model compared to the other nested one. In case of Multinomial Logistic Regression model, the comparison is carried out using the *likelihood ratio test*, which allows to compute the analysis of the deviances of the two models, where the deviance is equal to double log-likelihood. The difference of variances, called *Residual Deviance*, follows a χ^2 distribution

(even written as “Chi-squared distribution”). The p-value, which is the z test statistic applied to the Residual Deviance, helps to conclude whether there is an empirical evidence passing from the simpler model to the more sophisticated model. Two bidirectional hypothesis have to be compared, namely H_0 and H_1. H_0 is the hypothesis where the coefficients estimates are set to zero for those variables that not match between the simpler model variables and the simpler sophisticated variables, whereas H_1 is the opposite hypothesis of H_0 hypothesis. The first hypothesis H_0 suggests that both the models fit the data equally well. On the other hand, the second hypothesis H_1 suggests that the model with less variables outperforms the model with more variables in terms of data fit. H_0 is rejected when p-value < 0.05, otherwise H_0 is not rejected. The *anova* function suggests to pass from the *starting model* to the *updated model*, due to a p-value equal to 2.515e-12. For what concerns the metrics values of *starting model*, the Training Accuracy value is 0.827, the AIC value is 14407.1 and the BIC value is 19587.28. The result is that the interactions seen in the graphical evaluation of the dataset are useful for the model.

The final step is to find other interactions, if possible, to improve on the updated model. To this aim, all the pairs of quantitative variables with a correlation coefficients >= 0.5 have been checked. After that, the choice of the interaction terms is fallen in important correlations in the football domain which can improve the prediction of a player’s position. After some investigations, let *final model* be the *updated model* with the addition of *Short_Passing*:*Ball_Control*, *Aggression*:*Interceptions*, *Short_Passing*:*Long_Passing* and *Finishing*:*Long_Shots* interactions. Every predictor is significant, included the majority of all the factor levels. Moreover, the standard errors associated to every predictor are very satisfactory due to very low values. The choice of the *final model* with respect to the *updated model* is supported by the results of function *anova* (p-value equal to 0.048). For what concerns the metrics values of *final model*, the Training Accuracy value is 0.828 with a 95% confidence interval [0.822, 0.835], the AIC value is 14441.09 and the BIC value is 20012.24. AIC value and BIC value of *updated model* are smaller than the *final model*, but the difference of the values is very slight. No polynomials and natural splines have not been reported as significant for the improvement of the *final model*, which contains the variables listed in Figure 5.1.

After obtaining the coefficients estimates, *varImp* function by *R caret* package (see *Function varImp*) has been applied in order to examine what are the most important predictors. The *varImp* function returns all the covariates with the associated variable importance value, in which this value is the sum of the absolute values of the coefficients estimates for each level. Moreover, factors are treated as they was $M - 1$ independent covariates, in which M is



```
 [1] "Height_cm"                 "Weight_kg"                  "Preferred_Foot"
 [4] "Crossing"                  "Finishing"                  "Heading_Accuracy"
 [7] "Short_Passing"             "Volleys"                    "Dribbling"
[10] "Curve"                     "Free_Kick_Accuracy"         "Long_Passing"
[13] "Ball_Control"              "Acceleration"               "Sprint_Speed"
[16] "Agility"                   "Reactions"                  "Balance"
[19] "Shot_Power"                "Jumping"                    "Stamina"
[22] "Strength"                  "Long_Shots"                 "Aggression"
[25] "Interceptions"             "Positioning"                "Vision"
[28] "Penalties"                 "Composure"                  "Weak_Foot"
[31] "Attacking_Work_Rate"       "Defensive_Work_Rate"        "Body_Type"
[34] "Short_Passing:Ball_Control" "Aggression:Interceptions"  "Short_Passing:Long_Passing"
[37] "Finishing:Long_Shots"      "Acceleration:Sprint_Speed"  "Dribbling:Ball_Control"
[40] "Curve:Free_Kick_Accuracy"  "Long_Passing:Vision"        "Dribbling:Agility"
[43] "Ball_Control:Agility"
```

Figure 5.1: Variables of the final Multinomial Logistic Regression model

the number of the factor levels. For instance, *Preferred_Foot* factor has two levels which are Left, considered as the *baseline*, and Right; in the computation of the variable importance, *Preferred_FootRight* will be considered and its coefficients estimates will give a measure of the values difference between Right level and Left level. The variable importance values are reported in Table 5.1.

Table 5.1: Absolute sum of the coefficients estimates for every variable. AWR = Attacking_Work_Rate, DWR = Defensive_Work_Rate, BTL = Body_TypeLean, BTN = Body_TypeNormal, BTS = Body_TypeStocky, SP:BC = Short_Passing:Ball_Control, SP:LP = Short_Passing:Long_Passing, A:SS = Acceleration:Sprint_Speed, C:FKA = Curve:Free_Kick_Accuracy.

Height_cm	Weight_kg	Preferred_FootRight	Crossing
1.016	0.323	32.381	3.309
Finishing	Heading_Accuracy	Short_Passing	Volleys
2.204	4.432	4.410	0.583
Dribbling	Curve	Free_Kick_Accuracy	Long_Passing
1.925	0.497	0.560	2.413
Ball_Control	Acceleration	Sprint_Speed	Agility
3.864	4.936	4.359	1.597
Reactions	Balance	Shot_Power	Jumping
0.674	0.329	0.629	1.476
Stamina	Strength	Long_Shots	Aggression
1.799	3.451	1.410	2.970
Interceptions	Positioning	Vision	Penalties
7.998	2.055	2.181	0.416

Composure	Weak_Foot2	Weak_Foot3	Weak_Foot4
0.456	43.339	44.362	50.093
Weak_Foot5	AWRMedium	AWRHigh	DWRMedium
49.332	32.384	33.883	39.005
DWRHigh	BTL (170-185)	BTL (185+)	BTN (170-)
45.319	5.450	5.813	39.425
BTN (170-185)	BTN (185+)	BTS (170-)	BTS (170-185)
13.899	6.548	41.236	21.326
BTS (185+)	Body_TypeUnique	SP:BC	Aggression:Interceptions
96.334	10.099	0.048	0.008
SP:LP	Finishing:Long_Shots	A:SS	Dribbling:Ball_Control
0.033	0.016	0.055	0.039
Curve:FKA	Long_Passing:Vision	Dribbling:Agility	Ball_Control:Agility
0.008	0.019	0.034	0.401

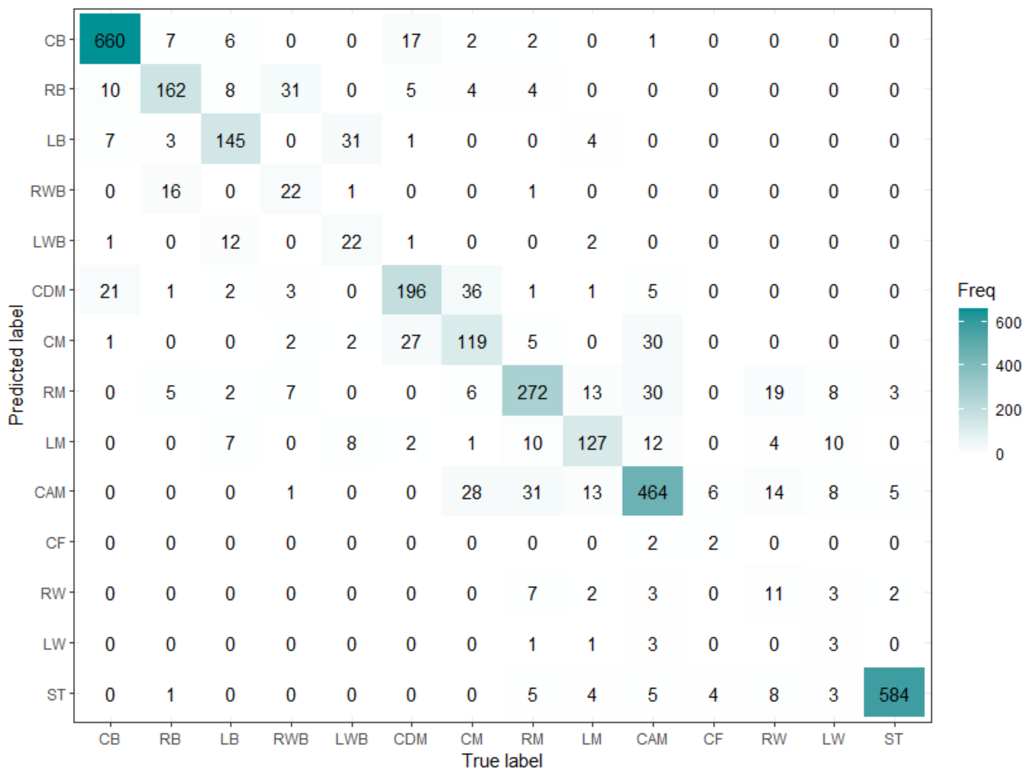
In general, the most important predictors are the following:

- *Preferred_FootRight*, with an overall value of 32.381;
- *Weak_Foot2*, with an overall value of 43.339;
- *Weak_Foot3*, with an overall value of 44.362;
- *Weak_Foot4*, with an overall value of 50.092;
- *Weak_Foot5*, with an overall value of 49.332;
- *Attacking_ Work_RateMedium*, with an overall value of 32.384;
- *Attacking_ Work_RateHigh*, with an overall value of 33.883;
- *Defensive_ Work_RateMedium*, with an overall value of 39.005;
- *Defensive_ Work_RateHigh*, with an overall value of 45.319;
- *Body_ TypeNormal (170-)*, with an overall value of 39.425;
- *Body_ TypeNormal (170-185)*, with an overall value of 13.899;
- *Body_ TypeNormal (185+)*, with an overall value of 6.548;

- *Body_ TypeStocky (170-)*, with an overall value of 41.236;
- *Body_ TypeStocky (170-185)*, with an overall value of 21.326;
- *Body_ TypeStocky (185+)*, with an overall value of 96.334;
- *Body_ TypeUnique*, with an overall value of 10.099.

A moderate contribute is given from all the remaining covariates. Moreover, all the interactions have the lowest contributes in the *final model*.
The code concerning Multinomial Logistic Regression is shown in Listing § A.3.

5.2.1 Predictions



	CB	RB	LB	RWB	LWB	CDM	CM	RM	LM	CAM	CF	RW	LW	ST
CB	660	7	6	0	0	17	2	2	0	1	0	0	0	0
RB	10	162	8	31	0	5	4	4	0	0	0	0	0	0
LB	7	3	145	0	31	1	0	0	4	0	0	0	0	0
RWB	0	16	0	22	1	0	0	1	0	0	0	0	0	0
LWB	1	0	12	0	22	1	0	0	2	0	0	0	0	0
CDM	21	1	2	3	0	196	36	1	1	5	0	0	0	0
CM	1	0	0	2	2	27	119	5	0	30	0	0	0	0
RM	0	5	2	7	0	0	6	272	13	30	0	19	8	3
LM	0	0	7	0	8	2	1	10	127	12	0	4	10	0
CAM	0	0	0	1	0	0	28	31	13	464	6	14	8	5
CF	0	0	0	0	0	0	0	0	0	2	2	0	0	0
RW	0	0	0	0	0	0	0	7	2	3	0	11	3	2
LW	0	0	0	0	0	0	0	1	1	3	0	0	3	0
ST	0	1	0	0	0	0	0	5	4	5	4	8	3	584

Figure 5.2: Confusion matrix in prediction phase for Multinomial Logistic Regression.

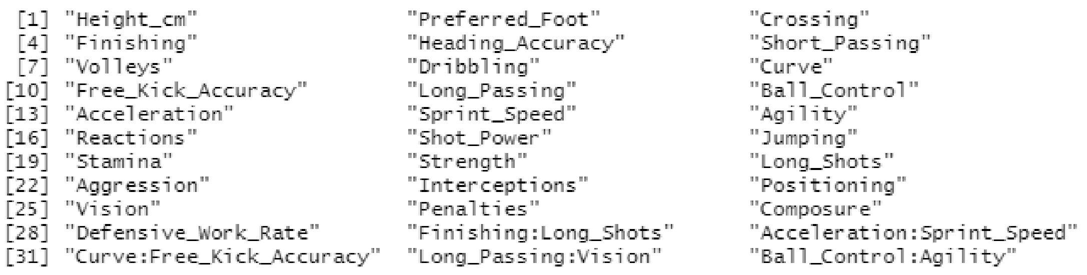
From the results obtained in training phase, we can suggest that the Multinomial Logistic Regression model is quite satisfactory due to the significance of all the variables in the model

and the good results in training metrics values. For what concerns the metrics values in prediction phase, the Test Accuracy value is 0.818 with a 95% confidence interval [0.805, 0.831] and the AUC value is 0.963.
The Test Accuracy value is slightly lower than the Training Accuracy value, but it is coherent with the behaviour held by the model after a training phase, in which the training error underestimates the test error. Sensitivity values, obtained computing the confusion matrix after the prediction phase, deserve attention. The sensitivity metric is computed for each response variable level and it represents the percentage of the correct predicted values identified. Sensitivity metric allows to identify what are the less accurate predicted player's positions. It follows that CB and ST are the best predicted classes, with a sensitivity value respectively of 0.943 and 0.983. On the other hand, the worst predicted positions are RWB, LWB, CF, RW and LW, with a sensitivity value respectively of 0.333, 0.344, 0.167, 0.196 and 0.086. These bad results are given mainly for the reason that the training set contains less observations for these positions. It is even curios to observe that, excluded CF position, the worst predicted positions involve the side area of a football pitch. The other positions not mentioned above have a sensitivity value bigger than 0.6, so the prediction is not aleatory. Figure 5.2 shows the confusion matrix obtained in prediction phase.
AUC value is extremely satisfactory due to its value really close to 1.

5.3 Automatic model selection

The number of variables found as significant in the *final model* is pretty high. For this reason, automatic model selection has been applied for choosing the best relevant variables of the model according to AIC metric and BIC metric. Backward selection strategy has been chosen because it is preferable to start from the full model when the number of variables is very high. Indeed, with a different selection strategy, i.e., forward selection, the risk of obtaining too few variables in the final model is high, with the consequence that the final model can explain an unsatisfactory amount of variability. The resulting model will be called *backward model*.
Automatic model selection has been performed with *step* function given by the *R stats* package (see *Function step*). By default, the variables are chosen by AIC metric due to the $k = 2$ parameter which is applied to the penalization term. The backward selection strategy has been applied two times with different metrics, one time with AIC metric and one time with BIC metric. To apply BIC metric, k parameter has been changed in $k = ln(N)$ as required.

From a theoretical point of view, BIC metric tends to penalize complex models more than AIC metric if $ln(N) > 2$, where N is the number of training set observations. This would be the case, because $ln(13608) = 9.518 > 2$. Nevertheless, the BIC penalization did not happen in practice, because both backward strategy with AIC metric and backward strategy with BIC metric return the same results in terms of coefficients estimates and metrics values. For this reason, whatever model can be evaluated. Our choice falls in backward selection with AIC metric.
First, the backward selection process has identified in *Weight_kg*, *Balance*, *Weak_Foot*, *Attacking_ Work_Rate* and *Body_ Type* the variables to be removed from the *final model*. In addition to it, the backward selection process has identified in *Short_Passing*:*Ball_ Control*, *Aggression*:*Interceptions*, *Short_Passing*:*Long_Passing*, *Dribbling*:*Ball_ Control* and *Agility*:*Dribbling* the interactions to be removed from the *final model*. All the covariates included in the *backward model* are significant and the standard errors are very satisfactory due to very low values. For what concerns the metrics values of the *backward model*, the Training Accuracy value is 0.824 with a 95% confidence interval [0.817, 0.830], the AIC value is 14265.7 and the BIC value is 17686.58.
Figure 5.3 shows the variables included in the *backward model*.



```
 [1] "Height_cm"                  "Preferred_Foot"           "Crossing"
 [4] "Finishing"                  "Heading_Accuracy"         "Short_Passing"
 [7] "Volleys"                    "Dribbling"                "Curve"
[10] "Free_Kick_Accuracy"         "Long_Passing"             "Ball_Control"
[13] "Acceleration"               "Sprint_Speed"             "Agility"
[16] "Reactions"                  "Shot_Power"               "Jumping"
[19] "Stamina"                    "Strength"                 "Long_Shots"
[22] "Aggression"                 "Interceptions"            "Positioning"
[25] "Vision"                     "Penalties"                "Composure"
[28] "Defensive_Work_Rate"        "Finishing:Long_Shots"     "Acceleration:Sprint_Speed"
[31] "Curve:Free_Kick_Accuracy"   "Long_Passing:Vision"      "Ball_Control:Agility"
```

Figure 5.3: Variables of the backward model.

The most important predictors, given by *varImp* function, are the following:

- *Preferred_FootRight*, with an overall value of 31.914;
- *Defensive_ Work_RateMedium*, with an overall value of 35.233;
- *Defensive_ Work_RateHigh*, with an overall value of 41.746.

A moderate contribute is given from all the remaining covariates. Moreover, all the interactions have the lowest contributes in the *backward model*.
The code concerning the automatic model selection is shown in Listing § A.4.

5.3.1 Predictions

From the results obtained in training phase, we can suggest that the *backward model* is quite satisfactory due to the significance of all the variables in the model and the good results in training metrics values. For what concerns the metrics values in prediction phase, the Test Accuracy value is 0.817 with a 95% confidence interval [0.803, 0.830] and the AUC value is 0.965. As expected, the Test Accuracy value is slightly lower than the Training Accuracy value. For what concerns the sensitivity values, they are really close to the sensitive values found in the Multinomial Logistic Regression model. Due to this fact, the problem in predicting positions involved in the side area of a football pitch still exists.
AUC value is slightly better than the one obtained with Multinomial Logistic Regression and it is extremely satisfactory.

5.4 Linear Discriminant Analysis

LDA is a different technique from Multinomial Logistic Regression and it has been performed with *lda* function given by the *R MASS* package (see *Function lda*). LDA algorithm aims at finding the $K - 1$ discriminant functions used to build a decision rule for assigning an observation to one class.
The application of the algorithm requires some assumptions to be satisfied. First, the response variable should have more than two levels with similar size. This assumption is partially respected due to the substantial difference in observations between some levels. A second assumption is related to the normal distribution which the quantitative covariates of the training set need to follow. To check the assumption, Shapiro-Wilk normality test could be applied, but the *R* function requires an input sample with a maximum of 5000 observations. To overcome this problem, Anderson-Darling normality test is suitable for larger sample size and can be applied through the *ad.test* function given by the *R nortest* package (see *Function ad.test*). For every quantitative variable is returned the associated test p-value. If the p-value is higher than 0.05, the distribution of the quantitative data is not significantly different from normal distribution; in other words, the normality can be assumed. All the resulting p-values are less than 0.05, so the quantitative variables do not follow a normal distribution. Failure to meet the assumptions may suggest that LDA will perform worse than Multinomial Logistic Regression. To confirm these suggestions, it is necessary to apply the *lda* function giving in input the same predictors included in the *final model*. The *lda* function computes

the prior probability of each level, the group mean of each level and the set of linear discriminant coefficients which allow to form the formulas for each linear discriminant function LDi, with $i = 1, ..., K - 1$. The canonical discrimination evaluation provides the amount of variance explained from each linear discriminant. $LD1$ explains more than half the variance, exactly the 54%. This value decreases by increasing i and the result is that $LD1$, $LD2$, $LD3$, $LD4$ and $LD5$ explain together the 91% of the total variance of the sample. For what concerns the metrics values obtained by LDA model, the Training Accuracy value is 0.742 with a 95% confidence interval [0.735, 0.750], while AIC metric and BIC metric cannot be computed. It is a matter of fact that LDA Training Accuracy is lower than the Training Accuracy obtained with Multinomial Logistic Regression and the automatic model selection process.
The code concerning LDA is shown in Listing § A.5.

5.4.1 Predictions

From the results obtained in training phase, we can suggest that the LDA model is not satisfactory due to the failure to meet the starting assumptions. In confirmation to this, the Test Accuracy value is 0.72 with a 95% confidence interval [0.703, 0.733] and the AUC value is 0.880. Both the test metric values are lower than the Test Accuracy and AUC obtained with Multinomial Logistic Regression and the automatic model selection process. Moreover, the graphical tool *ldahist* allows to understand whether the model obtained is satisfactory in discriminating each LDi.
The predictions provided by LDA are illustrated in Figure 5.4 and Figure 5.5. Predictions are plotted with histograms through the *ggplot2* package. There are $K - 1$ different histograms, one for each linear discriminant function. The plots are useful to understand whether each discriminant function separates each level well. If the histograms partially overlap, it means the groups are not well differentiated. Only the first two linear discriminant functions are considered, due to an acceptable explained deviance of the model equal to 77%. On each plot we can see that the separation of the levels is not so satisfactory. For example, for the $LDA1$ function RB, LB, RWB, LWB and CDM levels are grouped together and completely overlapped; even RM, LM and CAM are grouped together and finally the remaining forward positions CF, RW, LW and ST. So, with the first linear discriminant function only CB position is differentiated quite good, whereas the other positions are not singularly distinguishable. We can observe a similar behaviour for $LDA2$ and the main problem is that there are

too many response variable levels and it is hard to find a well-separation for each group.

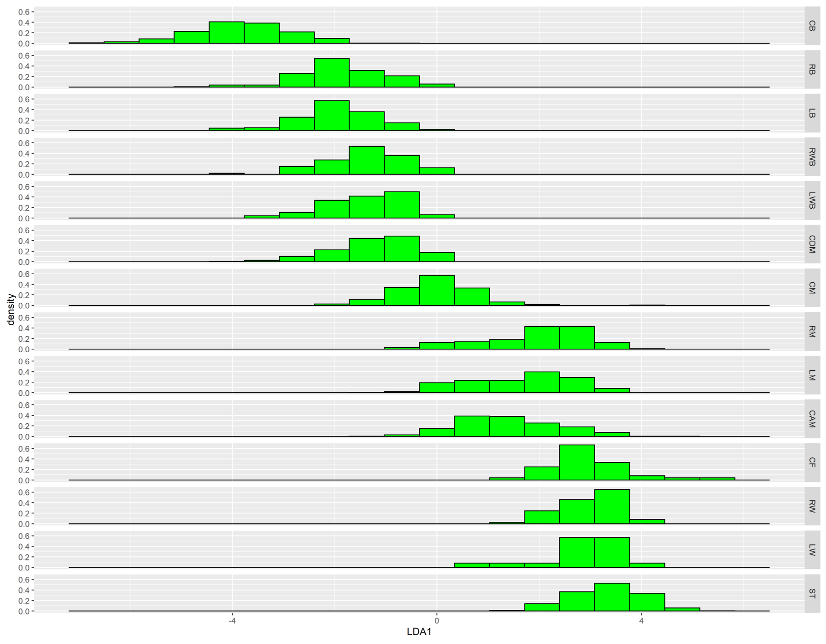


Figure 5.4: LDA1 predictions.

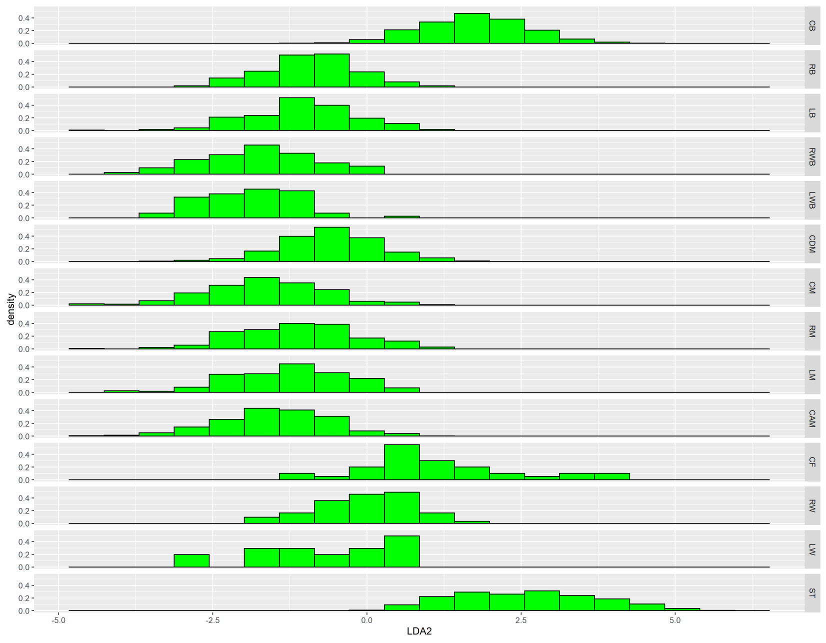


Figure 5.5: LDA2 predictions.

Even graphically, it is confirmed that LDA model performs worse than Multinomial Logistic Regression model. Due to the high number of levels in the response variable, the suggestion could be either relevel the response variable reducing the number of responses, or transform the data so that the normality assumption for the quantitative variables is true and that the levels of the response variable are more than two with similar size.

5.5 Regularization methods

With the exception of LDA, all the techniques analyzed until now are quite satisfactory. Anyway, the drawback is the large number of covariates inside these models with the consequence of being complex. Moreover, the majority of the quantitative covariates has high variance and a certain amount of anomalous observations can influence the performance of the models. For these reasons, shrinkage techniques, namely Ridge Regression and Lasso, have been applied in order to simplify the models reducing the variability of the estimators. Before starting with the analysis of techniques' results, it is necessary to create the data structure for both the training set and the test set. Indeed, two matrix structures have to be created starting from the *final model* formula, where each matrix contains the predictors (except the intercepts) as columns and the observations values as rows. If a predictor is a factor with M levels, $M - 1$ dummy variables will be created and each of them refer to the *baseline*. More in detail, *X_training* matrix is built referring to the training set observations and *X_test* matrix is built referring to the test set observations. Finally, the responses associated to each observation are created separately. In particular, *Y_training* is the response factor which refers to the *Best_Position* column of the training set, and *Y_test* is the response factor which refers to the *Best_Position* column of the test set. Both Ridge Regression and Lasso have been performed with *glmnet* function given by the *R glmnet* package (see *Function glmnet*). The *alpha* parameter is set to 0 for Ridge Regression implementation and set to 1 for Lasso implementation. Moreover, the *family* parameter is set as *multinomial* in order to fit a log-scaled logistic regression model for each response variable level. Then, the *type.measure* parameter is equal to *class* for obtaining misclassification error. Finally, k-Fold Cross-Validation (see Subsection § 7.1.2) has been used to tune the λ hyperparameter, with k=10 and through the *cv.glmnet* function given by the *R glmnet* package (see *Function cv.glmnet*). To make predictions, the *glmnet.fit* object from the *cv.glmnet* function is used, which is the fitted model using the best λ on the entire training set.

5.5.1 Ridge Regression

Ridge Regression is the first regularization method applied and it is responsible on shrinking the coefficients estimates close to zero. The suggestion is that Ridge Regression should not be too much better in terms of metrics values than Multinomial Logistic Regression, because Ridge Regression does not make variable selection and the complexity of the model is not so reduced.

The process starts by considering an automatic grid of λ values. Each λ value is associated to the percentage of explained deviance and the coefficients estimates are returned for each level. As expected, the number of covariates does not vary and the higher the $ln(\lambda)$ the more the coefficients estimates are close to 0. Moreover, the higher the $ln(\lambda)$ the smaller the explained deviance value is, with the consequence of obtaining less accurate models. To find the most accurate model, it is necessary to find the best λ value. The 10-Fold Cross-Validation technique has been applied to find the best λ value, according to the minimum misclassification error.

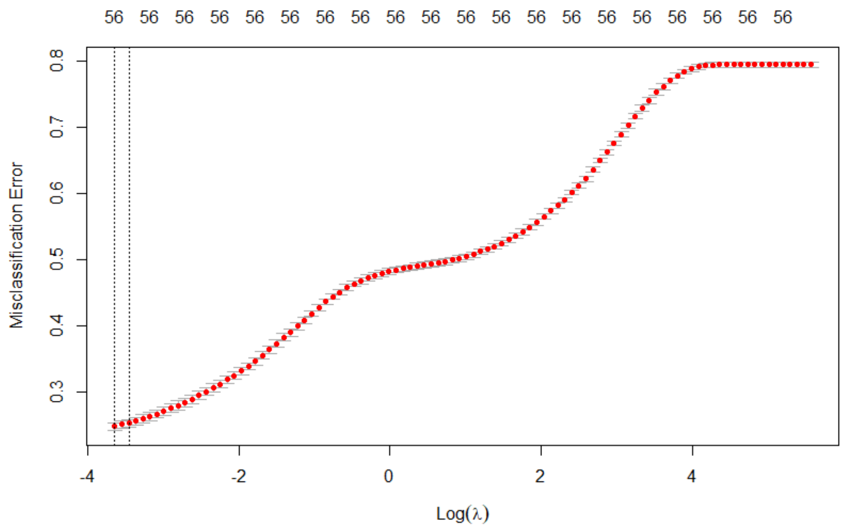


Figure 5.6: Misclassification error varying λ value during the 10-Fold cross validation with Ridge Regression. The leftmost vertical dashed line is the λ that corresponds to the minimum misclassification error, and the rightmost vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error.

Figure 5.6 shows that the higher the $ln(\lambda)$ the higher the misclassification error. Moreover, misclassification error increases rapidly from $ln(\lambda) = 0$. Then, there are two different λ values. The leftmost vertical dashed line is the λ that corresponds to the minimum misclassification error, and the rightmost vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error. The latter λ has a slight bigger misclassification error. In this scenario, variable selection has not been applied, so it is preferable to choice the λ that corresponds to the minimum misclassification error. In addition, the misclassification error is 0.25 and the best λ is 0.026. Figure 5.7 shows how the coefficients estimates decreases for ST level with the $ln(\lambda)$ variation, with the vertical dashed line that corresponds to the best λ. We can observe that the coefficients estimates, which corresponds to the best λ, are not quite shrunk.

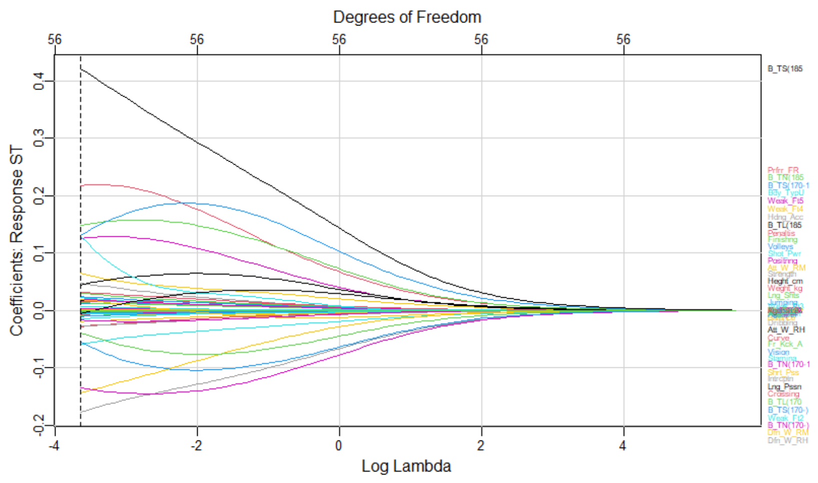


Figure 5.7: Coefficients estimates for ST level varying λ value with Ridge Regression. Each variable, associated to a specific color, is reported on the right of the plot.

Moreover, Figure 5.8 shows that the maximum explained deviance obtained from the best λ is 0.632. For what concerns the metrics values of final model, the Training Accuracy value is 0.757 with a 95% confidence interval [0.750, 0.764], the AIC value is -38773.36 and the BIC value is -38352.8. AIC value and BIC value are very satisfactory and they represent the lowest values of all the models seen until now. On the other hand, the Training Accuracy value is the worst value between all the models analyzed until now. Ridge Regression turns out to have worse performance even analyzing sensitivity values. While the majority of levels are quite satisfactory in sensitivity values, the worst predicted levels with a sensitivity value less than 1% are RWB, LWB, CF, RW and LW. It is the demonstration that Ridge Regression, besides not making variable selection, is the worst model according to the metrics values obtained in the training phase.

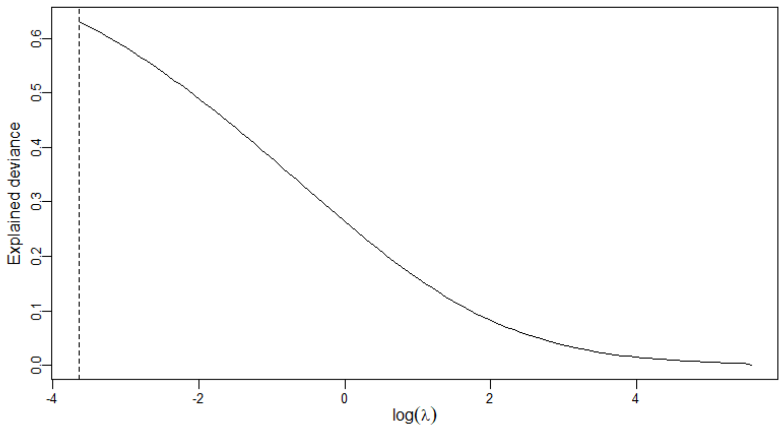


Figure 5.8: Explained deviance according to λ value in Ridge Regression. The vertical dashed line is the λ that corresponds to the minimum misclassification error.

The code concerning Ridge Regression is shown in Listing § A.6.

5.5.2 Lasso

Lasso is the other regularization method applied and it is responsible on shrinking the coefficients estimates close to zero and applying variable selection. The suggestion is that Lasso should be better in terms of metrics values than Multinomial Logistic Regression, because the number of variables will be reduced.

The process starts by considering an automatic grid of λ values. Each λ value is associated to the percentage of explained deviance and the coefficients estimates are returned for each level. As expected, the number of covariates varies. Indeed, the higher the $ln(\lambda)$ value the more variables are selected and the associated coefficients estimates set to 0. Moreover, the higher the $ln(\lambda)$ the smaller the explained deviance value is, with the consequence of obtaining less accurate models. To find the most accurate model, it is necessary to find the best λ value. The 10-Fold Cross-Validation technique has been applied to find the best λ value, according to the minimum misclassification error. Figure 5.9 shows that the higher the $ln(\lambda)$ the higher the misclassification error. Moreover, misclassification error increases rapidly from $ln(\lambda) = -4$. Then, there are two different λ values. The leftmost vertical dashed line is the λ that corresponds to the minimum misclassification error, and the rightmost vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error. The minimum misclassification error corresponds to 0.188 and the minimum misclassification error plus 1 standard error is equal to 0.191. The difference between the two latter error values is almost imperceptible, so the λ that corresponds to the minimum misclassification error plus 1 standard error has been preferred due to the highest variable selection, in which the variables maintained in the model are 32. The best λ has a value of 4.271e-04.

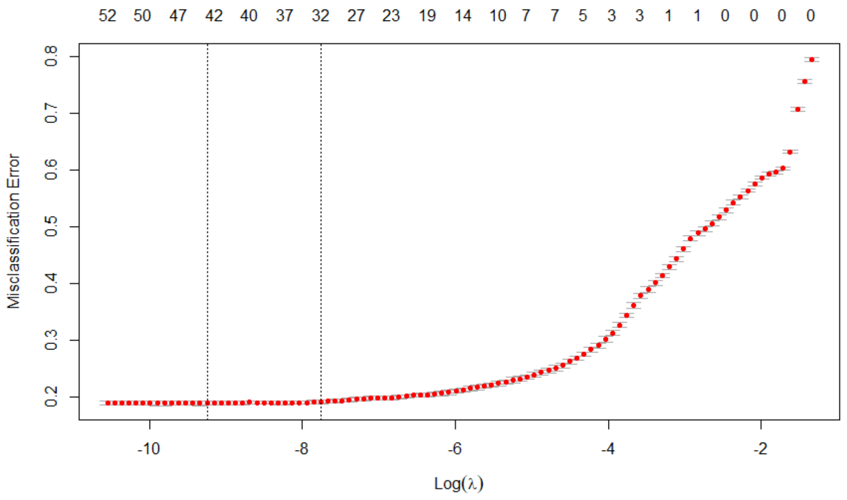


Figure 5.9: Misclassification error varying λ value during the 10-Fold cross validation with Lasso. The leftmost vertical dashed line is the λ that corresponds to the minimum misclassification error, and the rightmost vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error.

Figure 5.10 shows how the coefficients estimates decreases for ST level with the $ln(\lambda)$ variation, with the vertical dashed line that corresponds to the best λ. We can observe that the coefficients estimates, which corresponds to the best λ, are quite shrunk and some variables are no longer influential due to a coefficient estimate equal to 0. Moreover, Figure 5.11 shows that the maximum explained deviance obtained from the best λ is 0.7730992. For what concerns the metrics values of final model, the Training Accuracy value is 0.821 with a 95% confidence interval [0.814, 0.827], the AIC value is -47461.61 and the BIC value is -47041.05. AIC value and BIC value are very satisfactory and they are better than the values returned by Ridge Regression. Moreover, the Training Accuracy value is quite near to the best value obtained with Multinomial Logistic Regression. For what concerns the sensitivity values, the majority of levels are predicted very well, with a values range that goes from 0.7 to 0.95. Nevertheless, some levels as RWB, LWB, CF, RW and LW have unsatisfactory sensitivity, in particular the forward positions that have values less than 0.13. We can conclude saying that

Lasso performs similarly to Multinomial Logistic Regression in terms of metrics values, in which Lasso has sharply obtained better results for AIC metric and BIC metric but Lasso has the disadvantage of being inferior for what concerns the worst predicted sensitivity values. Finally, Lasso turns out to have satisfactory performance in the training phase thanks to the application of variable selection.

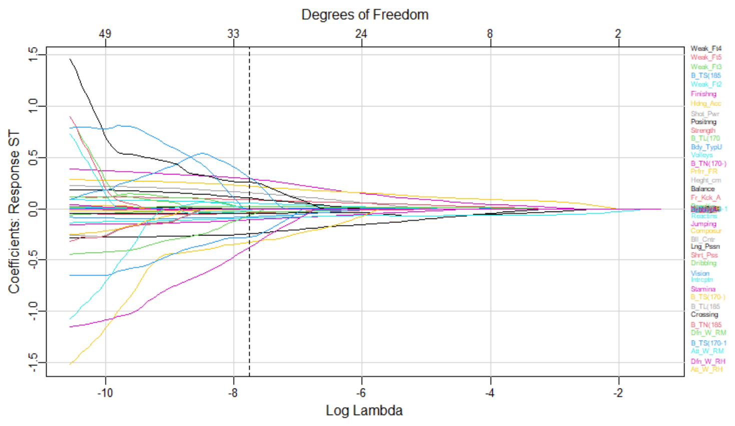


Figure 5.10: Coefficients estimates for ST level varying λ value with Lasso. Each variable, associated to a specific color, is reported on the right of the plot.

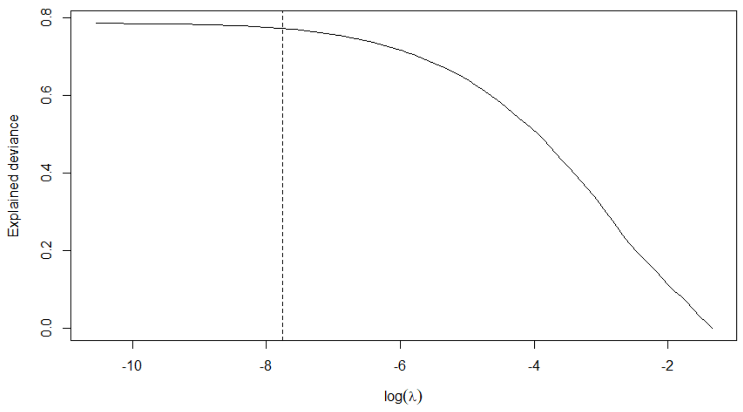


Figure 5.11: Explained deviance according to λ value in Lasso. The vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error.

The final step is to analyze the importance of predictors according to the coefficients estimates obtained. Due to the high number of levels and the associated covariates, only the coefficients estimates about ST and CB player's positions will be reported. The absolute shrunk coefficients estimates differ for each response variable level as suggested from Figure 5.12 and Figure 5.13. All the zero values represent the predictors chosen in the variable selection process which have no influence on the players' positions predictions. The absolute coefficients estimates for ST level assume values in the range between 0 and 0.377, with the result that the effect of the coefficients is not that punchy. Anyway, the most influential predictors are *Defensive_ Work_RateHigh*, *Attacking_ Work_RateHigh*, *Body_ TypeUnique* and *Finishing*. For what concerns the absolute coefficients estimates for CB level, their values are included in a range between 0 and 1.75. In this case, *Body_ TypeNormal (170-)* has a great impact on the resulting model than the other influential predictors as *Body_ TypeNormal (170-185)*, *Defensive_ Work_RateHigh*, *Heading_Accuracy* and *Preferred_FootRight*. Indeed, the difference of the absolute coefficients estimates between the highest value and the second highest value is around 1.3.

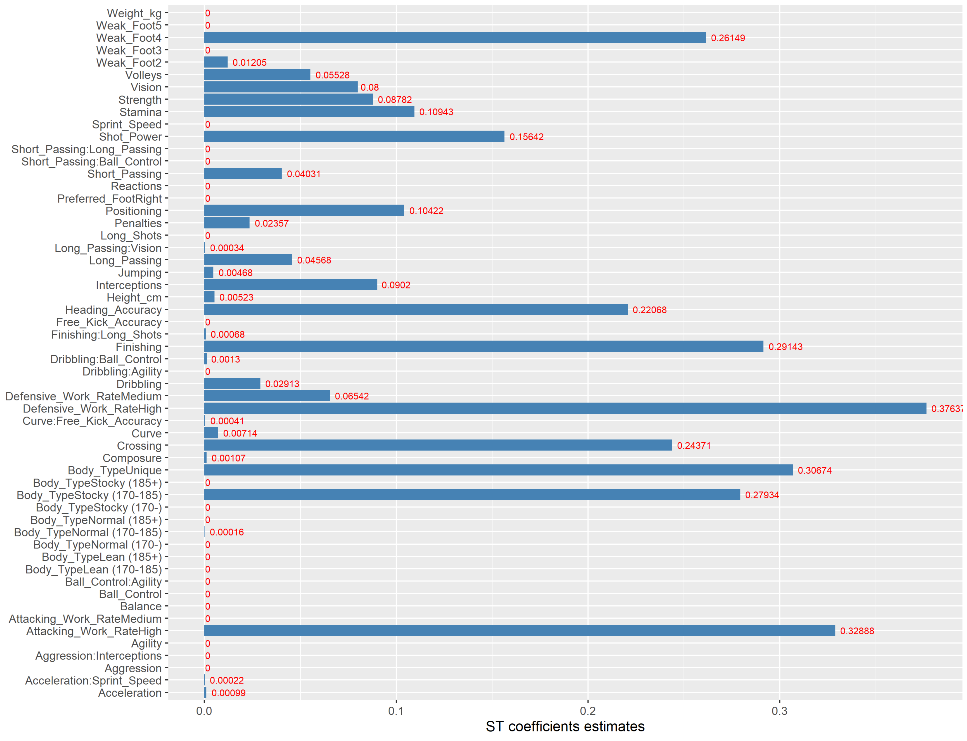


Figure 5.12: Absolute coefficients estimates for every variable of the ST level.

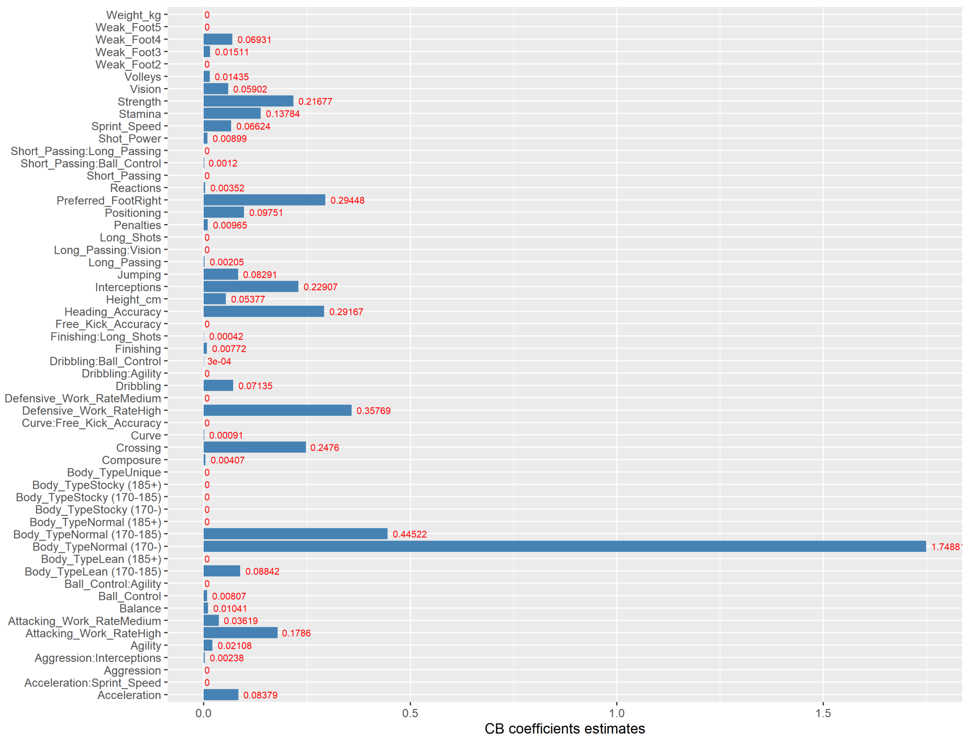


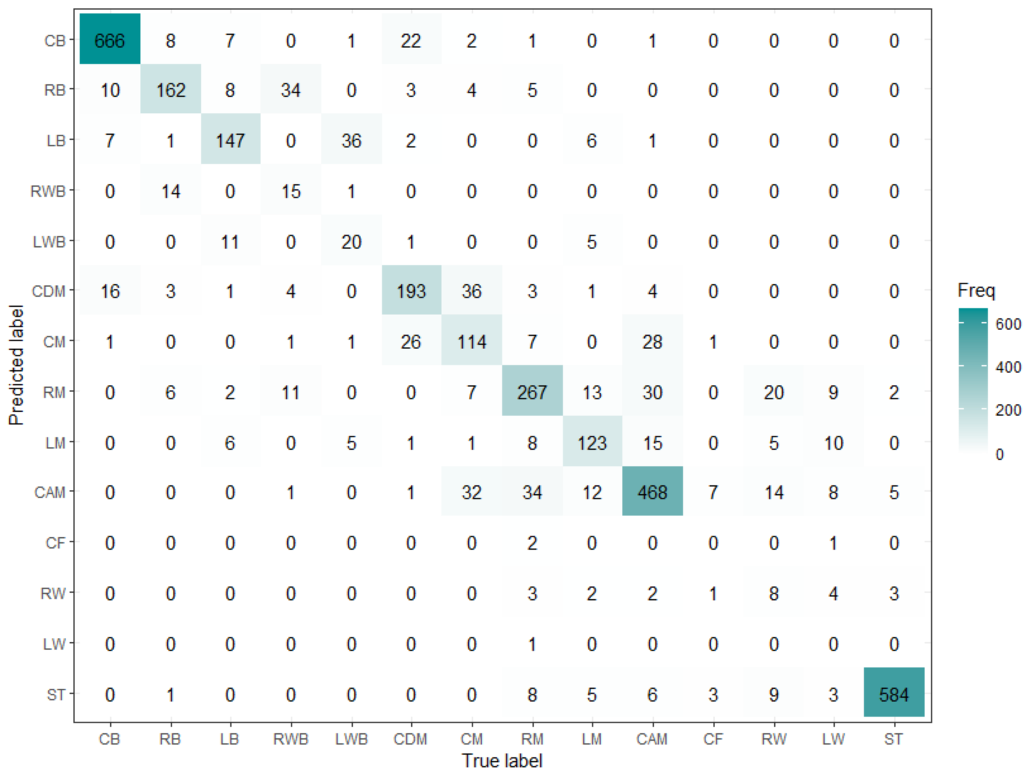
Figure 5.13: Absolute coefficients estimates for every variable of the CB level.

The code concerning Ridge Regression is shown in Listing § A.7.

5.5.3 Predictions

Given the Ridge Regression metrics values obtained in the training phase, we cannot expect better results in prediction phase. For what concerns the metrics values in prediction phase, the Test Accuracy value is 0.748 with a 95% confidence interval [0.733, 0.763] and the AUC value is 0.93. The Test Accuracy is slightly lower than the Training Accuracy, as expected. For what concerns the sensitivity values, the situation does not change for RWB, LWB, CF, RW and LW levels. CB and ST levels are the most accurate as verified in the previous methods. The only positive side is that the AUC value is very satisfactory.
Contrary to what was seen with Ridge Regression, the satisfactory results obtained during the training phase suggests that Lasso is expected to return satisfactory results even in the prediction phase. For what concerns the metrics values, the Test Accuracy value is 0.811 with a 95% confidence interval [0.798, 0.824] and the AUC value is 0.962. The Test Accuracy is slightly lower than the Training Accuracy, as expected. Moreover, the metrics values

obtained in prediction phase are almost equivalent to that obtained with Multinomial Logistic Regression. For what concerns the sensitivity values, the situation does not change too much for the worst predicted response variable levels. More in detail, RWB, LWB and RW levels have an higher sensitivity values with respect to those obtained in the training phase. Unfortunately, CF and LW levels have a sensitivity value less than 1% and they are predicted as midfielders with greater propensity to attack. The AUC value is very satisfactory. Figure 5.14 shows the confusion matrix obtained in prediction phase for Lasso.



	CB	RB	LB	RWB	LWB	CDM	CM	RM	LM	CAM	CF	RW	LW	ST
CB	666	8	7	0	1	22	2	1	0	1	0	0	0	0
RB	10	162	8	34	0	3	4	5	0	0	0	0	0	0
LB	7	1	147	0	36	2	0	0	6	1	0	0	0	0
RWB	0	14	0	15	1	0	0	0	0	0	0	0	0	0
LWB	0	0	11	0	20	1	0	0	5	0	0	0	0	0
CDM	16	3	1	4	0	193	36	3	1	4	0	0	0	0
CM	1	0	0	1	1	26	114	7	0	28	1	0	0	0
RM	0	6	2	11	0	0	7	267	13	30	0	20	9	2
LM	0	0	6	0	5	1	1	8	123	15	0	5	10	0
CAM	0	0	0	1	0	1	32	34	12	468	7	14	8	5
CF	0	0	0	0	0	0	0	2	0	0	0	0	1	0
RW	0	0	0	0	0	0	0	3	2	2	1	8	4	3
LW	0	0	0	0	0	0	0	1	0	0	0	0	0	0
ST	0	1	0	0	0	0	0	8	5	6	3	9	3	584

Figure 5.14: Confusion matrix in prediction phase for Lasso.

In conclusion, Lasso is one of the best technique found and it has similar results to those obtained with Multinomial Logistic Regression. On the other hand, Ridge Regression model is not suitable to our case.

6 Machine Learning Techniques

This chapter describes Machine Learning techniques which belong to the classification domain. The aim of Machine Learning methods is to make predictions of the categorical response variable *Best_Position*. The techniques involved in this chapter are Decision Tree, Random Forest, K-Nearest Neighbour, Naive Bayes and Support Vector Machine. Further on, a comparison between Machine Learning techniques and the other Data Mining techniques deepened in Chapter § 4 will be investigated.

6.1 Premises: Bagging

Bagging, acronym of Boostrap Aggregating (James et al., 2021, Chapter 8), is an ensemble technique in which t simple classifiers, called *weak learners*, are trained independently and in parallel. When the training phase comes to the end, a new instance can be classified through the voting technique, in which each classifier predicts the class (a.k.a. vote) and, after combining these predictions altogether, the most predicted class is returned. Figure 6.1 shows graphically Bagging steps.
The main assumption in ensemble techniques is that every *weak learner* is independent of each other. The Bootstrapping resampling method is the attempt to reach the independence, allowing the creation of t different samples for training each classifier. These samples have the same training set size, with the difference that the instances are sampled with replacement. From a theoretical point of view, building separate models with its own training set and

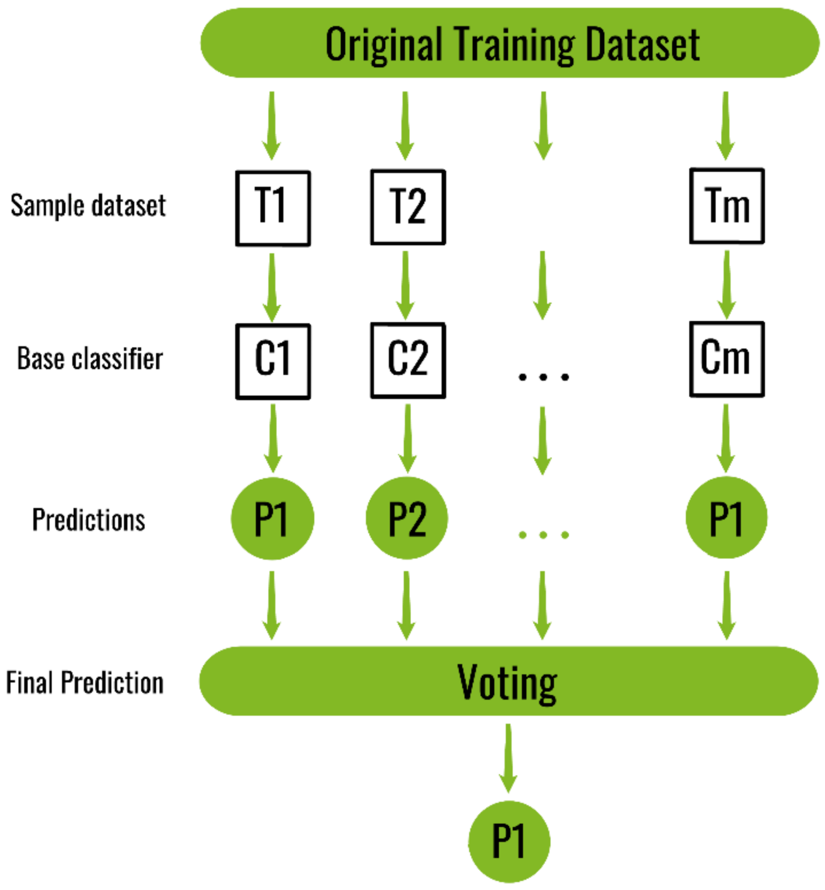


Figure 6.1: How Bagging technique works.
Source: https://www.geeksforgeeks.org/ml-bagging-classifier/

averaging the prediction results leads to reduce the variance and to maintain the same bias. In practice, variance can be reduced but samples are not completely independent, with the consequence that the bias tends however to increase. So, Bagging does not work well when the samples are too much similar or when the algorithms used for classifiers are stable.

This technique is very useful for Decision Trees (see Section § 6.2), which suffers of high variance. Combining them, the prediction values improves due to the reduction of the variance, and the result of this aggregation is the Random Forest technique (see Section § 6.3). Even though the interpretation of Bagging technique is difficult, it is possible to obtain a summary of variable importance taking the mean of the sum of all the Gini index entropy measures obtained from each classifier.

6.2 Decision Tree

Decision Tree (Mitchell, 1997, Chapter 3) is a supervised learning technique which allows to learn if-then rules inferred by data predictors and representable through a tree structure, as in Figure 6.2. Every decision tree is composed by nodes and branches, where each node represents an attribute and a branch represents one of the possible values of the attribute itself. A decision tree starts with the starting node, called root, represented by a specific attribute and from which m branches are created downwards, where m is the number of possible values that the attribute can assume. From this point two alternatives are available:

- another decision tree can start from a branch, which is called *subtree*, and this process continues until reaching a leaf node, which corresponds to a specific class of the response variable;
- if there are no other attributes to explore, a leaf node is returned.

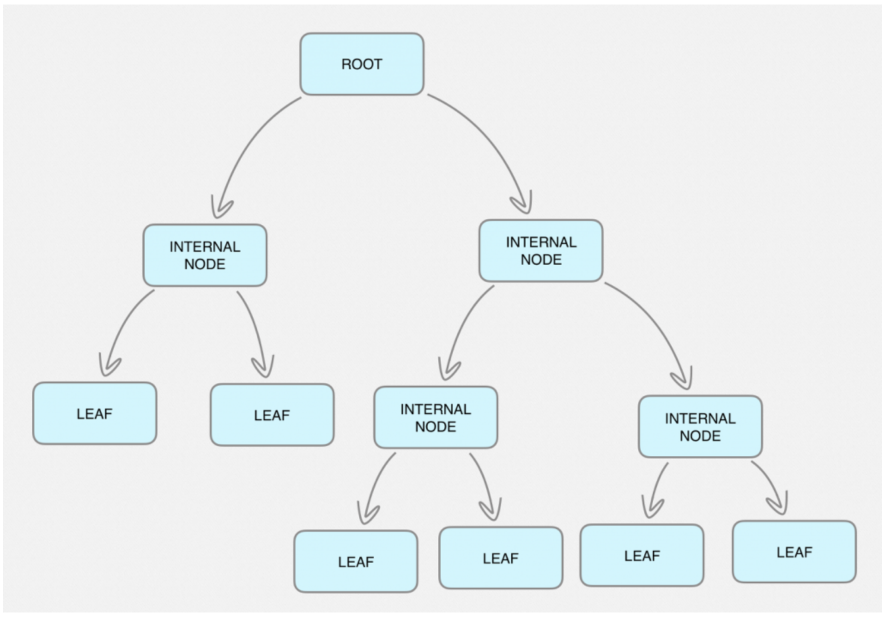


Figure 6.2: The basic structure of a decision tree.
Source: https://blog.quantinsti.com/decision-tree/

In this context, a decision tree is called *classification tree* and it can be represented as a disjunction of conjunction of constraints that defines all the possible paths from the root to the leaf node for a specific response variable class.

ID3[G] (Iterative Dichotomiser 3) algorithm allows to build decision trees with top-down recursive approach in which the core of the algorithm is to select the optimal attribute to test at each node of the tree. To this aim, the attribute which maximises the *information gain* is chosen, where *information gain* is a statistical property which measures the expected impurity reduction obtained by partitioning the training set instances with the attribute itself (Mitchell, 1997, Chapter 3). Formula (6.1) defines the *information gain* as

$$G(S, a) = I(S) - \sum_{v \in V(a)} \frac{|S_{a=v}|}{|S|} I(S_{a=v}), \tag{6.1}$$

where S represents the training set instances, a is the attribute involved, $V(a)$ is the set of values that the attribute a can assume and finally I is the impurity function. More in detail, the impurity function I measures the quality of a split for a certain attribute. Let $prop_k = \frac{|S_k|}{|S|}$ be the proportion of training set instances labelled with class k, with $k = 1, ..., K$. Then, two possible impurity functions are (see *Decision Trees mathematical formulation*)

- *Entropy*, which computes the Shannon entropy of the possible classes, formulated as $I(S) = -\sum_{k=1}^{K} prop_k \ln(prop_k)$;
- *Gini Index*, formulated as $I(S) = \sum_{k=1}^{K} prop_k(1 - prop_k)$.

The deeper the tree, the more complex the if-then rules, with the consequence that the model fits better. Nevertheless, if limits on the decision tree depth are not set, the decision tree learns perfectly the training data at the expense of not predicting well unseen data. In other words, the risk of overfitting is very high. To support this, ID3 algorithm assumption is that shorter decision trees are preferred over larger trees and the attributes with highest *information gain* are close to the decision tree root. This assumption suggests to limit the depth of the decision tree if the overfitting phenomenon occurs, or to apply pruning techniques as Reduced Error Pruning or Rule Post-Pruning.

6.3 Random Forest

Random Forest (James et al., 2021, Chapter 8) is a supervised learning technique which takes inspiration by Bagging method. With Random Forest t different Decision Trees are built, each of these trained by a different training set sample created with the Bootstrapping technique. The main difference from the Bagging method concerns the split operation when

the optimal attribute needs to be chosen for every node. While in Decision Tree technique the entire set of attributes is considered, in Random Forest only m attributes are chosen randomly, with $m < p$ and p is the total number of available attributes. When the training phase comes to the end, predictions from each Decision Tree are aggregated to return the class through the voting technique.
The choice of taking randomly only a subset of attributes at each node is made for respecting the independence assumption between all the Decision Trees involved. Indeed, if every Decision Tree chooses a strong variable from the training set due to the highest *information gain*, all the Decision Trees would be correlated and so very similar averaging them. It is clear that this choice would be useless and it would broke the independence assumption. Indeed, the Random Forest aim is to build different Decision Trees that are uncorrelated between them and random subsets of features are considered at each node to reach the goal.
According to the literature (Hastie, Tibshirani, and Friedman, 2009, Chapter 15), Random Forest technique should outperform Decision Tree technique in terms of metrics values. It is important the choice of the m parameter, that is the cardinality of the variables subset. It is suggested to use a small value if the variables are highly correlated. Moreover, the number of Decision Trees (indicated with the t parameter) is suggested to be as high as possible in order to not incur in overfitting.

6.4 K-Nearest Neighbour

K-Nearest Neighbour, well-known with *KNN*[G] acronym (Mitchell, 1997, Chapter 8), is a supervised learning and instance-based technique for approximating discrete-valued target functions. The learning process simply saves every training data instance $\langle x, f(x) \rangle$ into the memory, where x is the instance itself and $f(x)$ is the target function. The classification of a new instance x_{new} depends by the parameter K, a positive integer which allows to identify the closest training data instances to x_{new}. The new instance x_{new} is classified with the most common class between all the K nearest training data instances. In this thesis, K is the number of response variable levels, so to not confound the mathematical notation we assume from now that the parameter K will be named T.
The main assumption is that each instance can be viewed as a point in an Euclidean p-dimensional space, where $p = 1, ..., P$ and p is the number of predictors, and the Euclidean distance determines the T nearest training data instances. Let $a_p(x_{new})$ be the p-th feature of the new instance x_{new}. Then, the Euclidean distance between two different instances x_i and

x_j is

$$d(x_i, x_j) = \sqrt{\sum_{p=1}^{P} \{a_p(x_i) - a_p(x_j)\}^2}.$$

T nearest training data instances have been found and the new instance x_{new} is classified as follows:

$$f(x_{new}) = \arg\max_{k \in K} \sum_{t=1}^{T} I\{k = f(x_t)\},$$

where $k = 1, ..., K$ denotes the number of response variable levels, and the identity function $I\{k = f(x_t)\}$ is equal to 1 when the equality is verified (otherwise 0). The classification algorithm can be refined adding a weight for each of the T nearest neighbors, in order to make neighbors closer to the new instance x_{new} more relevant. The weights $(w_1, ..., w_T)$ are determined by a kernel function of the Euclidean distance, which corresponds to the inverse square of the distance between each neighbour and the new instance, as follows:

$$w_t = K(d(x_t, x_{new})) = d(x_t, x_{new})^{-2},$$

with $t = 1, ..., T$. Consequently, the new instance x_{new} is classified as follows:

$$f(x_{new}) = \arg\max_{k \in K} \sum_{t=1}^{T} w_i I\{k = f(x_t)\}.$$

If the points exactly matches with the consequence that $d(x_t, x_{new})^2$ is equal to 0, then $f(x_{new}) = f(x_t)$.

The KNN algorithm is considered as robust to noisy data and it works well with a large training set. Unfortunately, it is not possible to compute the variable importance. Choosing the right T parameter is crucial in the KNN performance; indeed, a small T value leads to obtain a flexible classifier with low bias and high variance, on the other hand a big T value leads to obtain a less flexible classifier with high bias and low variance. By a computational point of view, KNN is slower in prediction phase due to the entire set of instances to consider. To speed up the memory indexing process, *kd-tree* method allows to store every instance at a tree leaves. In this way, the closer the instances to each other, the closer to each other in the tree leaves.

6.5 Naive Bayes

Naive Bayes (Mitchell, 1997, Chapter 6) is a supervised learning technique based on Bayes' theorem. The training set is composed by tuples composed by an instance x and its corresponding target value, where the instance x is represented as a conjunction of attribute values $x_1, ..., x_p$ and the target function can assume one of the possible K values. Each training set instance is used in the training phase to compute the probability terms of the Bayes formula and, when the training phase is end, all the attribute values are given in input to the Naive Bayes classifier, returning at the end the most likely class. The Bayesian formulation is

$$\begin{aligned} P(Y = k|X = x_1, ..., x_p) &= \arg\max_{k \in K} \frac{P(x_1, ..., x_p|k)P(k)}{P(x_1, ..., x_p)} \\ &= \arg\max_{k \in K} P(x_1, ..., x_p|k)P(k), \end{aligned} \tag{6.2}$$

where

- $P(k)$ is the prior probability of falling into class k;
- $P(x_1, ..., x_p|k)$ is the posterior probability of observing the attribute values of instance x given the target class k;
- $P(x_1, ..., x_p)$ is the prior probability of observing the attribute values of instance x.

Formula (6.2) is simplified removing the denominator $P(x_1, ..., x_p)$ because it does not depend by k, so it becomes a constant and irrelevant for the classification. Anyway, the term *Naive* is applied to this technique because the main assumption is that each attribute value of the instance x is conditionally independent, given the target value of the instance x. With the classic Bayes formula, $K * N$ different combinations are necessary to compute $P(x_1, ..., x_p|k)$. Instead, if the main assumption is applied, then the number of different combinations to compute $P(x_1, ..., x_p|k)$ is simply $\prod_{p=1}^{P} P(x_p|k)$. The final results is that the assumption allows to transform an intractable problem to be computationally feasible, allowing a drastic reducing of the training computational time. Even though the assumption is not always respected, it has been demonstrated that Naive Bayes technique could perform well. So, Naive Bayes assigns a class to a new instance x according to probability

$$P(Y = k|x_1, ..., x_p) = \arg\max_{k \in K} P(k) \prod_{p=1}^{P} P(x_p|k).$$

So, before predicting the class of a new instance, $P(k)$ and $P(x_p|k)$ are the estimated probabilities computed starting from the training set instances. Moreover, $P(x_p|k)$ is equal to $\frac{n_k}{total_{n_k}}$, where n_k is the number of training set instances of class k for which x_p attribute value is encountered and $total_{n_k}$ is the total number of training set instances of class k. Anyway, this computation can lead to obtain a result equal to 0 if there are no instances n_k. As workaround, *m-estimate of probability* is provided and it is equal to $\frac{n_k+mp}{total_{n_k}+m}$, where

- p value is the prior probability, usually fixed as $1/k$ if the attribute contains k values;
- m value, namely *equivalent sample size*, is a constant used to weight the prior probability.

Parameters p and m allow to increase the size of the original sample with the addition of virtual instances, in order to avoid zero probability.

6.6 Support Vector Machine

Support Vector Machine, well-known with *SVM*[G] acronym (James et al., 2021, Chapter 9), is a supervised learning technique which classifies new instances defining nonlinear boundaries for separating each response variable class. As shown in Figure 6.3, SVM can be thought of as a generalization of the *maximal margin classifier*, which defines the hyperplane in the *p*-dimensional space that divides the space in *p-1* parts.
The *p*-dimensional space contains each training set instance and the goal of the separating hyperplane is to be as far away as possible from the training set instances. Moreover, the distance between the hyperplane and the nearest training set observations is called *margin*, which gives rise to the maximal margin hyperplane. The nearest training set observations are called *support vectors* and the assumption is that they are the only observations which affect the change of the maximal margin hyperplane. Unfortunately, the *maximal margin classifier* cannon exist due to the impossibility of separating the *p*-dimensional space with linear boundaries, causing the non-existence of a separating hyperplane. For this reason, Support

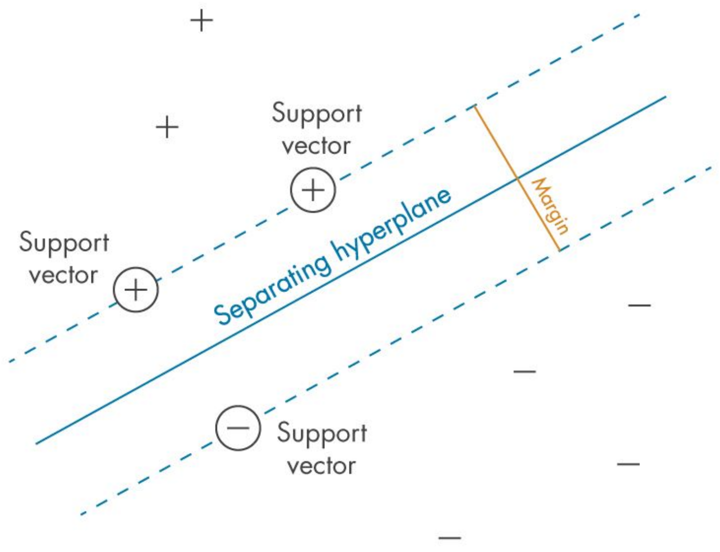


Figure 6.3: The basics of the Support Vector Machine.
Source: https://it.mathworks.com/discovery/support-vector-machine.html

Vector Classifier comes to our aid. Instead of having tiny *margins*, which make the maximal margin hyperplane change drastically and incur in overfitting, Support Vector Classifier relaxes the assumption in which all the training observations have to be classified correctly. Indeed, the presence of some training observations in the wrong hyperplane side (these are considered as misclassified) or in the wrong *margin* side is tolerated. The optimization problem for a linear classifier can be formulated as follows:

$$\begin{aligned} &\max_{\beta_0,\beta_1,\dots,\beta_p,\epsilon_0,\epsilon_1,\dots,\epsilon_n,M} \quad M, \\ &s.t. \quad y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M(1-\epsilon_i), \\ &\qquad\quad \sum_{p=1}^{P} \beta_p^2 = 1, \\ &\qquad\quad \epsilon_i \geq 0, \\ &\qquad\quad \sum_{n=1}^{N} \epsilon_n \leq C, \end{aligned} \tag{6.3}$$

where

- M is the hyperplane *margin* width;

- C is a nonnegative tuning parameter responsible of how much the *margin* errors are tolerated. If $C = 0$ no violations are tolerated, otherwise no more than C misclassified training set instances are accepted. The higher the C parameter the higher the bias and the lower the variance, conversely the lower the C parameter the lower the bias and the higher the variance;
- ϵ_i indicates the position of the i-th training set instance in the *p*-dimensional space. If $\epsilon_i = 0$, then the i-th observation is located in the right *margin* side, otherwise it is located in the wrong *margin* side. Moreover, if $\epsilon_i > 1$, then the i-th observation is located in the wrong side of the hyperplane, and it is considered as misclassified.

SVM extends the Support Vector Classifier formula (6.3) using a kernel function K, a similarity function which is applied to each feature in order to obtain nonlinear boundaries for discriminating every class in the *p*-dimensional space. This type of approach is called *kernel trick*. Moreover, the kernel function is computationally efficient because only $\binom{n}{2}$ inner products among all the pairs of the training set instances are computed. The optimization problem is reformulated as follows:

$$\begin{aligned}
&\max_{\beta_0,\beta_1,\dots,\beta_p,\epsilon_0,\epsilon_1,\dots,\epsilon_n,M} \quad M,\\
&s.t. \quad y_i(\beta_0 + \sum_{p=1}^{P} \beta_p K(x_i, x_{i'})) \geq M(1-\epsilon_i),\\
&\sum_{p=1}^{P} \beta_p^2 = 1,\\
&\epsilon_i \geq 0,\\
&\sum_{n=1}^{N} \epsilon_n \leq C.
\end{aligned}$$

The kernel function $K(x_i, x_i')$ takes in input two different training set observations, where $i \neq i'$, and they can be of different types as follows:

- *linear* kernel, which is $K(x_i, x_i') = \sum_{p=1}^{P} x_{ip} x_{i'p}$;
- *polynomial* kernel, which is $K(x_i, x_i') = (1 + \sum_{p=1}^{P} x_{ip} x_{i'p})^d$, where *d* is the nonnegative integer degree. The higher the degree the more flexible the decision boundaries;
- *rbf* kernel, which is $K(x_i, x_i') = \exp(-\gamma \sum_{p=1}^{P} (x_{ip} - x_{i'p})^2)$, where γ is a positive constant and it defines how much a single training set instance is influential. The larger the γ value, the closer other training set instances have to be in order to be influenced.

Finally, a one-vs-one classification approach is applied when the response variable classes are K > 2. The one-vs-one classification approach builds $\binom{K}{2}$ SVMs, in which every SVM compares two different classes k and k', with $k \neq k'$. Then, every SVM takes in input a test set instance in order to return the predicted class. At the end, the final classification corresponds to the most predicted class.

7 Machine Learning Results

This chapter focuses on the implementation and the discussion of the prediction results obtained from the Machine Learning techniques described in Chapter § 6. The details about the implementation will be supported by Python (Van Rossum and Drake, 2009) code.

7.1 Premises

7.1.1 Dataset adaptation

The same dataset used for Data Mining techniques has been exploited for Machine Learning techniques. Nevertheless, some modifications are needed for making the dataset compatible with Python data structures. The training set is composed by the 80% of the dataset observations and the test set is composed by the remaining 20% of dataset observations, as in Data Mining techniques (see Subsection § 5.1.1). For both of two dataset splits, observations are balanced such that the same multinomial distribution of the response variable is maintained. From this point, a set of operations has been applied both for training set and for test set.

The first operation is to ensure that the non-numeric attributes are considered as categorical variables, namely, *Best_Position*, *Weak_Foot*, *Attacking_ Work_Rate* and *Defensive_ Work_-Rate* variables. Next, it is necessary to transform every categorical variable into an one-hot encoding form, since algorithms like KNN can only work with numeric variables. One-hot encoding is applied with Python *sklearn.preprocessing.OneHotEncoder* library (see *Function*

OneHotEncoder) and it allows to create a number of new columns equal to the total number of different discrete values of the categorical variable involved. After that, the significant interactions found in Data Mining techniques are added. Each interaction is added as a column named *Variable1:Variable2*, in which every cell is the product between *Variable1* value and *Variable2* value. This step is necessary because, despite of *R* (R Core Team, 2022) methods, Python is not able to automatically add an interaction between two variables as column in a dataset. Afterwards, the predictors and the response variable of both the training set and the test set are split into two different data structures X and Y, compatible with the data structures required from the Machine Learning Python methods.

The final step is to create a scaled version of the training set and the test set through the so-called *standardization*, which is an attribute scaling technique that transform data so that to ensure mean 0 and standard deviation 1. *Standardization* is applied with Python *sklearn.preprocessing.StandardScaler* library (see *Function StandardScaler*) and its usage is suggested when a standard normal distribution for the attributes values is supposed. If the latter suggestion is not true, the alternative attribute scaling technique is *normalization*, which scales attributes values in a fixed range, usually between 0 and 1. Anyway, even though the attributes do not follow a normal distribution, standardized dataset has led to better results than the normalized dataset. So, the two standardized dataset splits will be used in KNN and SVM techniques, because they are not scale invariant.

The code concerning the dataset adaptation for Python is shown in Listing § B.7.

7.1.2 k-Fold Cross-Validation

All the Machine Learning techniques that we will see further on (except Naive Bayes) need to find the best hyperparameters, in order to obtain a model as accurate as possible and which does not suffer from the variability of the data and the overfitting phenomenon, when possible. Resampling methods comes to our aid, with samples from the training set drawn repeatedly and the model involved refitted on each sample. In this thesis, k-Fold Cross-Validation is the chosen cross validation approach, in which the training set is divided into k folds of equal size. One fold at a time, starting from the first to the last, is used for testing the model fitted on the remaining k-1 folds. The fold delegated for testing the fitted model is called *validation test*. The procedure is repeated k times, so that all folds are used as *validation test* one at a time. Finally, the average of the misclassification errors obtained at each step is returned as the estimate of the k-Fold Cross Validation (James et al., 2021, Chapter 5).

In this thesis, *GridSearchCV* is the function used for applying k-Fold Cross-Validation method within a set of hyperparameters, available from the Python *sklearn.model_selection* library (see *Function GridSearchCV*). There are three fundamental parameters to set in *GridSearchCV* function:

- *estimator*, which is the Machine Learning model we want to apply;
- *param_grid*, which is a dictionary composed by a set of hyperparameters to try in the estimator;
- *cv*, which determines the number of folds to generate from the training set. More in detail, *StratifiedKFold* function of the Python *sklearn.model_selection* library (see *Function StratifiedKFold*) has been applied, which allows to preserve the same multinomial distribution of the response variable on each sample.

We decided to apply a 10-Fold Cross-Validation, so that k=10 folds are generated. *GridSearchCV* function returns the estimator with the best found parameters refitted on the whole dataset.
The code concerning the application of the k-Fold Cross-Validation is shown in Listing § B.8.

7.1.3 Metrics

The metric used to evaluate a model after the training phase is **Training Accuracy**, which is the ratio between the sum of the training set correct predicted values for each response variable level and the total number of training set observations. The correct predicted values can be accounted for by the confusion matrix diagonal computed using the *confusion_matrix* function of the Python *sklearn.metrics* library (see *Function confusion_matrix*).
The metrics used for evaluating a model in prediction phase are listed below.

- **Test Accuracy**, which is the ratio between the sum of the test set correct predicted values for each response variable level and the total number of test set observations. The Test Accuracy computation is performed as in *Training Accuracy* metric.
- **AUC**, the same AUC metric described in Subsection § 5.1.2. The only difference is in the computation of the AUC value, obtained by the Python *sklearn.metrics.roc_auc_score* library (see *Function roc_auc_score*).

The code concerning the metrics computed is shown in Listing § B.9.

7.2 Decision Tree

Decision Tree is the first Machine Learning technique we consider, applied using the *DecisionTreeClassifier* function, available from the Python *sklearn.tree* library (see *Function DecisionTreeClassifier*). *DecisionTreeClassifier* function works only with numeric variables. Moreover, it allows to generate a decision tree classifier with the aim to predict the right class by learning decision rules deduced from the attributes of the training set. In a first trial, we build the default decision tree classifier from the training set, in which Gini Index impurity function is applied together with a maximum depth of the tree set to None. After the training phase, a perfect Training Accuracy equal to 1.0 is returned, as expected from the theory. Next, we make predictions on the fitted model, giving in input the test set and obtaining a Test Accuracy equal to 0.627. It is even possible to define a ranking of the variable importance, which is defined as the total reduction of the impurity function led by a specific feature. In other words, the most important variables are those with the higher *information gain*, and so the first nodes at the top of the decision tree. Figure 7.1 shows that *Positioning*, *Heading_Accuracy* and *Interceptions* are the first three variables selected for the decision tree building.

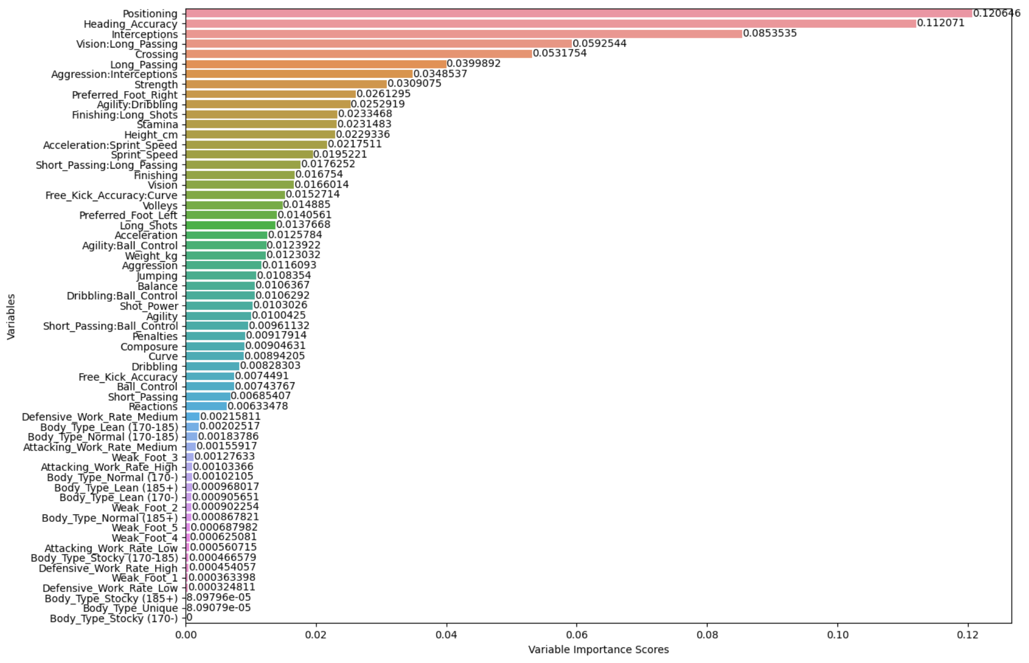


Figure 7.1: Variable importance of the standard decision tree.

Anyway, it is clear that the standard decision tree overfits. To limit the overfitting phenomenon, 10-Fold Cross-Validation has been applied in order to look for the best hyperparameters to set, which are:

- *criterion*, the impurity function considering *gini* and *entropy* values;
- *max_depth*, the maximum depth of the decision tree considering numeric values from 1 to 30.

After having applied 10-Fold Cross-Validation, the best model found has *criterion* hyperparameter set to *entropy* and *max_depth* hyperparameter set to 11. Limiting the maximum depth of the decision tree, we suggest that the Training Accuracy is lower than the Training Accuracy obtained with the standard decision tree. In addition to it, we expect that the Test Accuracy increases in order to get closer to the Training Accuracy. Indeed, the Training Accuracy is 0.846 and it is quite satisfactory. For what concerns the prediction phase, the Test Accuracy is set to 0.655, so it is closer to the Training Accuracy but anyway not very satisfactory. Finally, the returned AUC value is set to 0.816 and we can suggest that the overall class-specific performance is quite good. For what concerns the sensitivity values, it follows that the best predictions are obtained for CB and ST positions, with a sensitivity value of 0.88 and 0.87, respectively. The worst prediction is obtained for RWB, RW and LW positions, with a sensitivity value of 0.14, 0.11, and 0.03. respectively. These results confirm the trend to better predict the positions that correspond to the most frequent instances and central zones of the football pitch. As in the standard decision tree, Figure 7.2 shows that *Interceptions*, *Heading_Accuracy* and *Positioning* are the first three variables selected for the decision tree building, but with different values of variable importance.
The code concerning the Decision Tree technique is shown in Listing § B.10.

7.2.1 Pruning

Even though 10-Fold Cross-Validation has been applied, the difference between Training Accuracy value and Test Accuracy value is still large, supposing that the model overfits. To make the decision tree to generalize better, the *DecisionTreeClassifier* function offers a method called *cost_complexity_pruning_path* which computes the pruning path when Minimal Cost-Complexity Pruning algorithm is applied. More in detail, Minimal Cost-Complexity Pruning (see *Minimal Cost-Complexity Pruning*) is the pruning algorithm for decision trees in

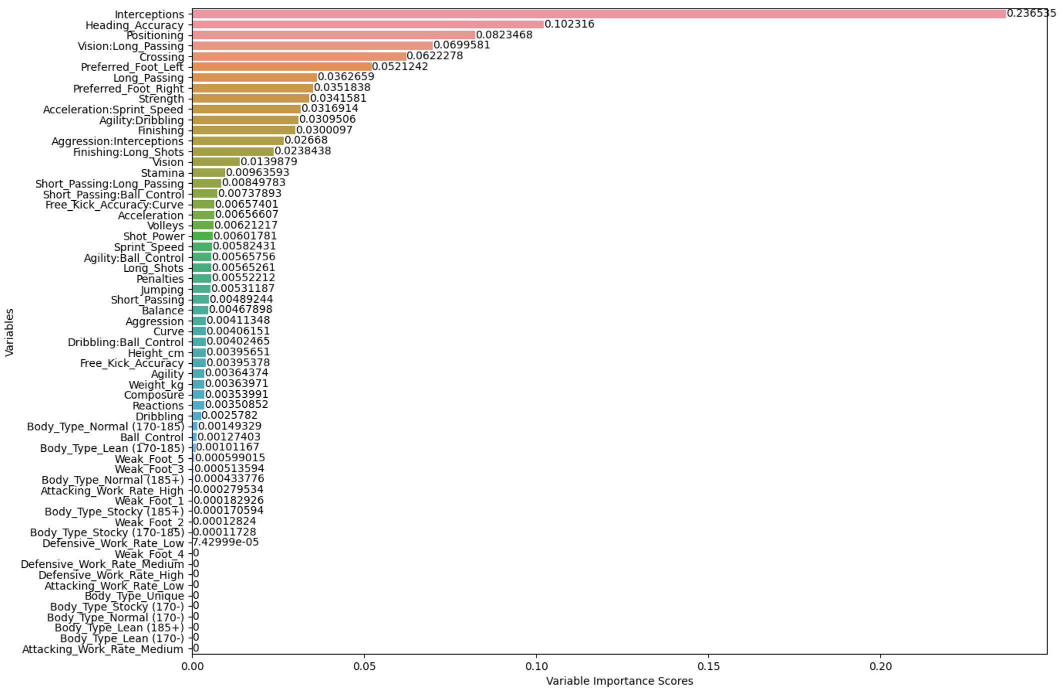


Figure 7.2: Variable importance of the decision tree after 10-Fold Cross-Validation.

which

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|,$$

where $R_\alpha(T)$ is the Cost-Complexity measure of the tree T to be minimized, $R(T)$ is the impurity function value of the leaves of tree T, $\tilde{T}$ is the cardinality of the leaves in tree T and finally α is the non-negative complexity hyperparameter that determines the pruning of nodes from the original tree T. A node of the decision tree is pruned when its effective α is the smallest. The pruning process can continue until reaching the parameter *ccp_alpha*, fixed before instantiating the *DecisionTreeClassifier* method. Now, the first step is to find the α values and the impurity values for each node at every step of the pruning process. A decision tree is trained for each α value. Figure 7.3 shows that the higher the effective α the higher the total impurity of the leaves of the tree (or the information gain of the leaves of the tree decreases). Moreover, the higher the α value the more the tree is pruned, with the consequence of reducing the number of nodes and the depth of the decision tree.

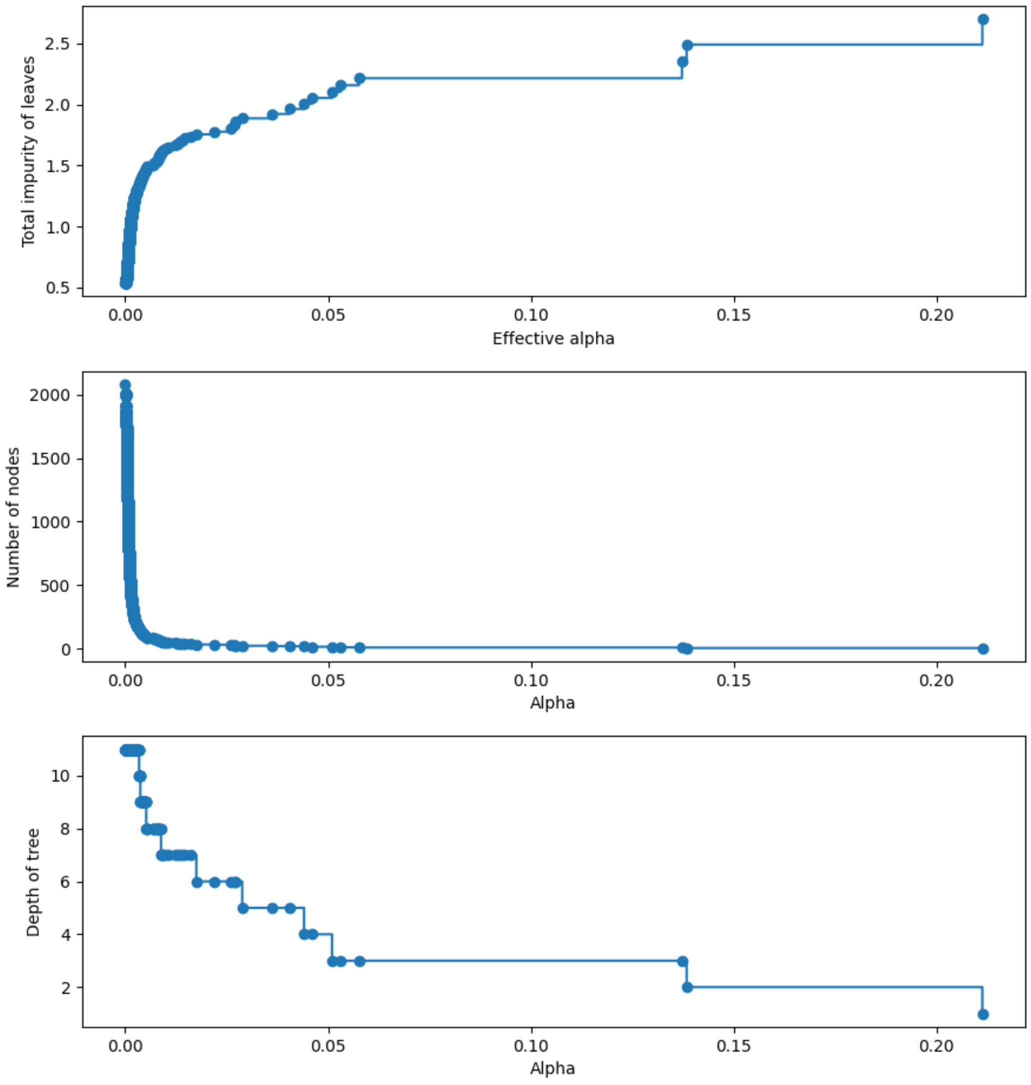


Figure 7.3: Total impurity of leaves, number of nodes, and tree depth in the pruned decision tree given α value.

Next, Figure 7.4 shows the evolution of Training Accuracy value and Test Accuracy value, according to the α value. It is clear that both the values decrease with the increase of α value, due to the increase of decision tree pruning. Moreover, the higher accuracy values are between $\alpha = 0$ and $\alpha = 0.03$. Finally, we can observe that the difference between the Training Accuracy value and Test Accuracy value is very small, so we can assume that the pruned decision tree generalizes well and it does not overfit. To confirm this assumption, we need to train a new decision tree fixing the *ccp_alpha* parameter to 1.277e-03, which corresponds to the α value that corresponds to the highest Test Accuracy value.

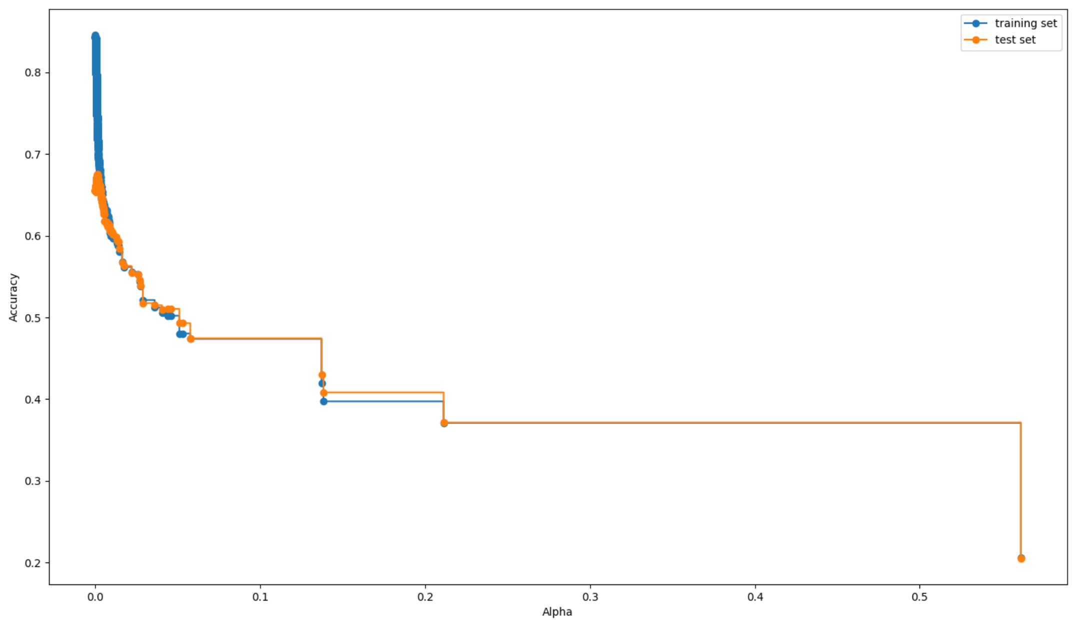


Figure 7.4: Training Accuracy value vs Test Accuracy value, given α value.

The Training Accuracy value is equal to 0.731 and it is a reasonable result, even though it is less that the Training Accuracy found after 10-Fold Cross-Validation. For what concerns the prediction phase, the Test Accuracy value is equal to 0.675 and it is the highest values found until now for this technique. Moreover, the difference between the Training Accuracy value and the Test Accuracy value is less than 0.05, confirming that the Minimal Cost-Complexity Pruning algorithm allows the model to not overfit. Finally, the AUC value is equal to 0.907, which is very satisfactory. For what concerns the sensitivity values, it follows that ST, CB, RB and LB positions are the best predicted, with a sensitivity value of 0.93, 0.89, 0.73, and 0.68, respectively. As seen with 10-Fold Cross-Validation, RWB, RW, LW and CF positions are those predicted worst, with sensitivity values that are less than 0.05. These results confirm the trend to better predict the positions that correspond to the most frequent instances and central zones of the football pitch. A little improvement is obtained for what concerns the central positions which flank CB player. As in the previous cases, Figure 7.5 shows that *Interceptions*, *Heading_Accuracy* and *Positioning* are the first three variables selected for the decision tree building, with 26 leaf nodes pruned.

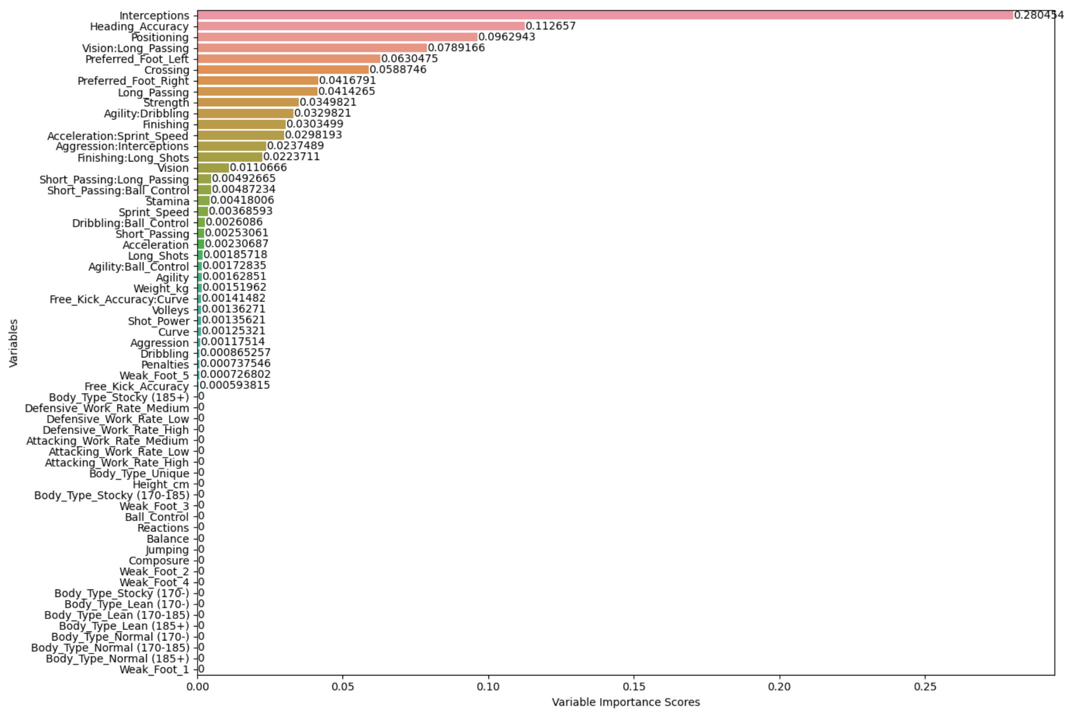


Figure 7.5: Variable importance of the decision tree after pruning.

At the end, we can conclude that Pruning technique allows to improve the model in terms of generalization, even though the metrics values are not so high and so not very reliable for predicting the position of a footballer, except for AUC value.
The code concerning the Decision Tree technique with the application of the pruning algorithm is shown in Listing § B.11.

7.3 Random Forest

As seen in Section § 7.2, the final results of a standalone decision tree could not be very satisfactory for prediction due to the high variability of the data, even pruning the decision tree itself. Random Forest technique comes to our aid, as it is composed by a fixed number of independent decision trees trained on a random subset of the original attributes. This technique allows to combine these decision tree to make a prediction and, on average, improving the performance with respect to a standalone decision tree. Random forest tends to achieve a low variance by the effect of the combination of the decision trees, at the cost of a slight increase of the bias. For these reasons, random forest technique should outperform decision tree technique.

Random Forest is applied using the *RandomForestClassifier* function, available from the Python *sklearn.ensemble* library (see *Function RandomForestClassifier*). In a first trial, we build the default random forest classifier from the training set, in which the number of estimators (a.k.a decision trees) is set to 100, the Gini Index criterion is applied as impurity function, the maximum depth of the independent decision trees is set to None and finally the maximum number of attributes on each subset is the square root of the number of attributes in the original training set. After the training phase, a perfect Training Accuracy equal to 1.0 is returned, as expected from the theory. Next, we make predictions on the fitted model, giving in input the test set and obtaining a Test Accuracy equal to 0.749. As in decision trees, it is possible to define a ranking of the variable importance in the random forest. Figure 7.6 shows that *Interceptions*, *Heading_Accuracy* and *Aggression:Interceptions* are the first three most important variables on average. We can notice that the variable importance scores are smaller than those obtained in decision tree classifier, but this behaviour is coherent with the fact that only a subset of variables is selected and it randomly changes for each decision tree.

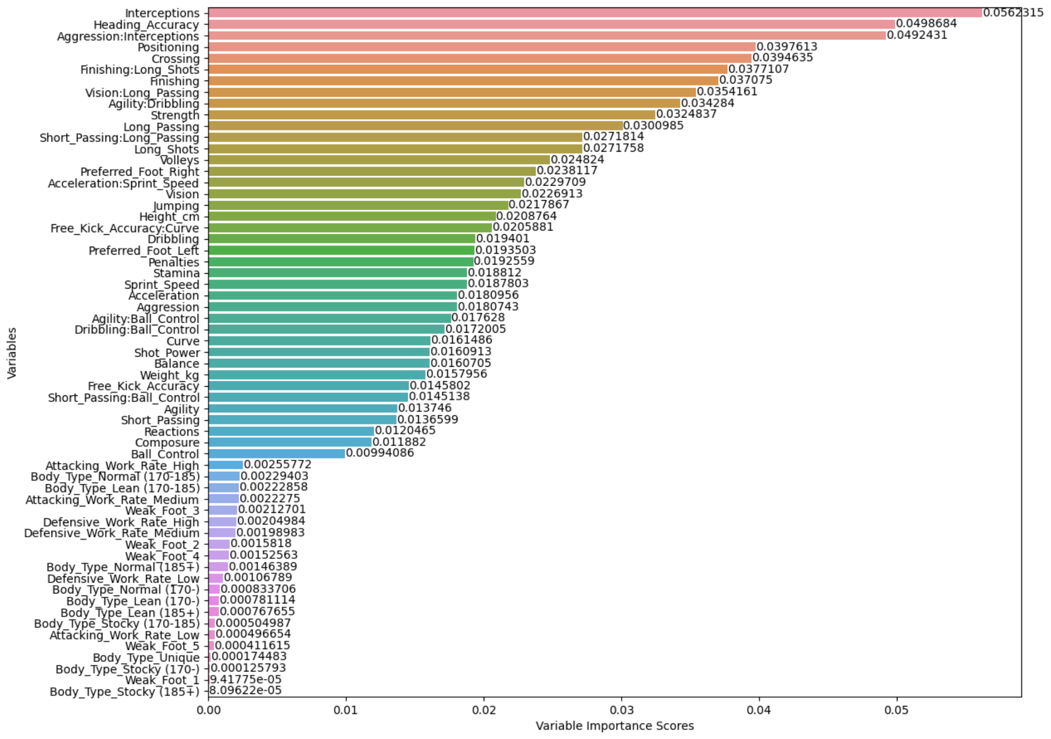


Figure 7.6: Variable importance of the random forest.

From the results obtained from the standard random forest, it is clear that the model overfits. To limit the overfitting phenomenon, 10-Fold Cross-Validation has been applied in order to look for the best hyperparameters to set, which are:

- *n_estimators*, the number of decision trees in the random forest, considering numeric values from 1 to 100;
- *criterion*, the impurity function considering *gini* and *entropy* values;
- *max_depth*, the maximum depth of each decision tree in the random forest considering numeric values from 1 to 30;
- *max_features*, the maximum number of attributes to consider from the training set for choosing the best attribute. The two possible choice are *sqrt* and *log2*, which corresponds respectively to the square root or the base 2 logarithm of the number of attributes of the original dataset.

After having applied 10-Fold Cross-Validation, the best model found has *n_estimators* hyperparameter set to *97*, *criterion* hyperparameter set to *entropy*, *max_depth* hyperparameter set to 28 and *max_features* hyperparameter set to *sqrt*. Limiting the maximum depth of each decision tree and reducing the number of attributes considered for the best split, we could obtain that the Training Accuracy is lower than the Training Accuracy obtained with the standard decision tree. In addition to it, we expect that the Test Accuracy increases in order to get closer to the Training Accuracy. Nevertheless, these assumptions do not come true drastically. Indeed, the Training Accuracy value remains unchanged to 1.0 as in the standard random forest. For what concerns the prediction phase, the Test Accuracy value is set to 0.753, so it is slightly closer to the Training Accuracy. The Test Accuracy value obtained is better than the Test Accuracy value of decision tree technique. Anyway, the difference between Training Accuracy value and Test Accuracy value is around 0.25, so the model tends to overfit. Finally, the returned AUC value is set to 0.951 and we can suggest that the overall class-specific performance is almost perfect. For what concerns the sensitivity values, it follows that ST, CB, LB, CAM, RB and CDM positions are the best predicted, with a sensitivity value of 0.96, 0.94, 0.83, 0.81, 0.77, and 0.71, respectively. CF, LW and RW are the worst predicted positions, with a sensitivity value less than 0.05. These results confirm that the central positions are the best predicted, in particular all the central defensive roles, the less and the more advanced central midfielder and the striker. As in the standard random forest, Figure 7.7 shows that *Interceptions*, *Aggression:Interceptions* and *Heading_Accuracy* are the first three variables selected for the decision tree building, even though with slight different values of variable importance.

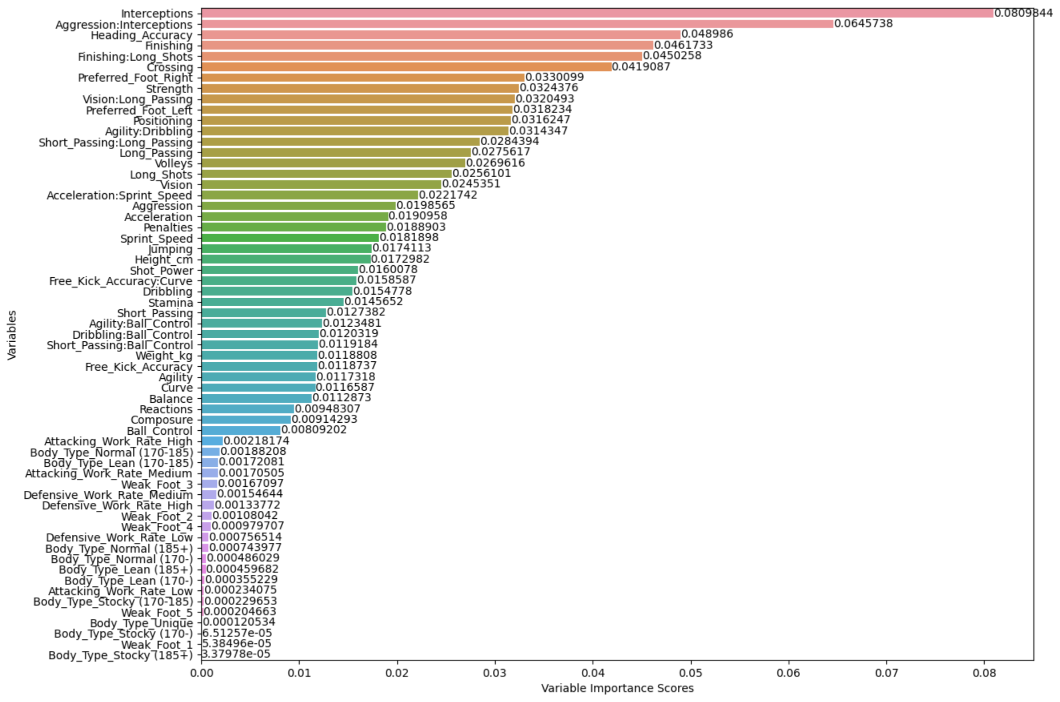


Figure 7.7: Variable importance of the random forest after 10-Fold Cross-Validation.

A final consideration is that Random Forest technique has been already built for improving the performance of decision trees selecting a random subset of variables for the best split. Indeed, there is no necessity of pruning decision trees in random forest, because random forest make the performance good with full depth due to the bootstrapping; moreover, the pruning technique is available only for standalone decision trees. So, checking *max_depth* and *n_estimators* parameters is enough.
The code concerning the Random Forest technique is shown in Listing § B.12.

7.4 K-Nearest Neighbour

KNN is a different technique in which the K nearest training set instances influence the prediction of an observation, and it is applied using the *KNeighborsClassifier* function available from the Python *sklearn.neighbors* library (see *Function KNeighborsClassifier*). The standardized version of training set and test set is used because KNN technique is not scale invariant. KNN is known to be robust to noisy data and to work well with large training data. In a first trial, we build the default KNNs classifier from the standardized training set, in which the number of neighbors to consider is set to 5 and the uniform weight function is applied. After the training phase, the Training Accuracy value is equal to 0.756, so the value is quite good. Next, we make predictions on the fitted model, giving in input the standardized test set and obtaining a Test Accuracy value equal to 0.637. The Test Accuracy value is not very satisfying and, in addition to it, the model could overfit due to a difference between the Training Accuracy value and the Test Accuracy value around 0.12. To improve the model, 10-Fold Cross-Validation has been applied in order to look for the best hyperparameters to set, which are:

- *n_neighbors*, the number of K neighbours used for classifying a new instance. The smaller the *n_neighbors* value the more flexible the classifier with low bias and high variance, at the contrary the higher the *n_neighbors* value the less flexible the classifier with high bias and low variance. A number of neighbors between 1 and 100 is considered;
- *weights*, the different types of weights applied for computing the nearest neighbours. In this case, *uniform* and *distance* values are considered, where *uniform* weights equally function all the neighbours instances and *distance* is the inverse square of the distance between each neighbour and the new instance. Note that *p* value, which determines the power of the distance metric parameter, is equal to 2 and it means that the Euclidean distance is computed.

After having applied 10-Fold Cross-Validation, the best model found has *n_neighbors* hyperparameter set to 98 and *weights* hyperparameter set to *distance*. We expect that the Test Accuracy value increases and that the model could generalize better. Surprisingly, the Training Accuracy value is equal to 1.0, with a perfect model fitting. Indeed, the Test Accuracy value is equal to 0.664 and it is slightly higher than the Test accuracy value found in the standard KNN classifier, but anyway not so satisfactory. Finally, the returned AUC value is set to 0.928 and we can suggest that the overall class-specific performance is very satisfactory. For what concerns the sensitivity values, it follows that ST, CB, CAM and LB position are the best predicted, with a sensitivity value of 0.90, 0.89, 0.83, and 0.7,5 respectively. Unfortunately there are many positions as CF, LW, RW, LWB and RWB which have sensitivity values lower than 0.05. These results confirm the trend to better predict the positions that correspond to the most frequent instances and central zones of the football pitch.

Finally, the results obtained suggests that the model overfits, and all the lateral positions are badly predicted.
The code concerning the KNN technique is shown in Listing § B.13.

7.5 Naive Bayes

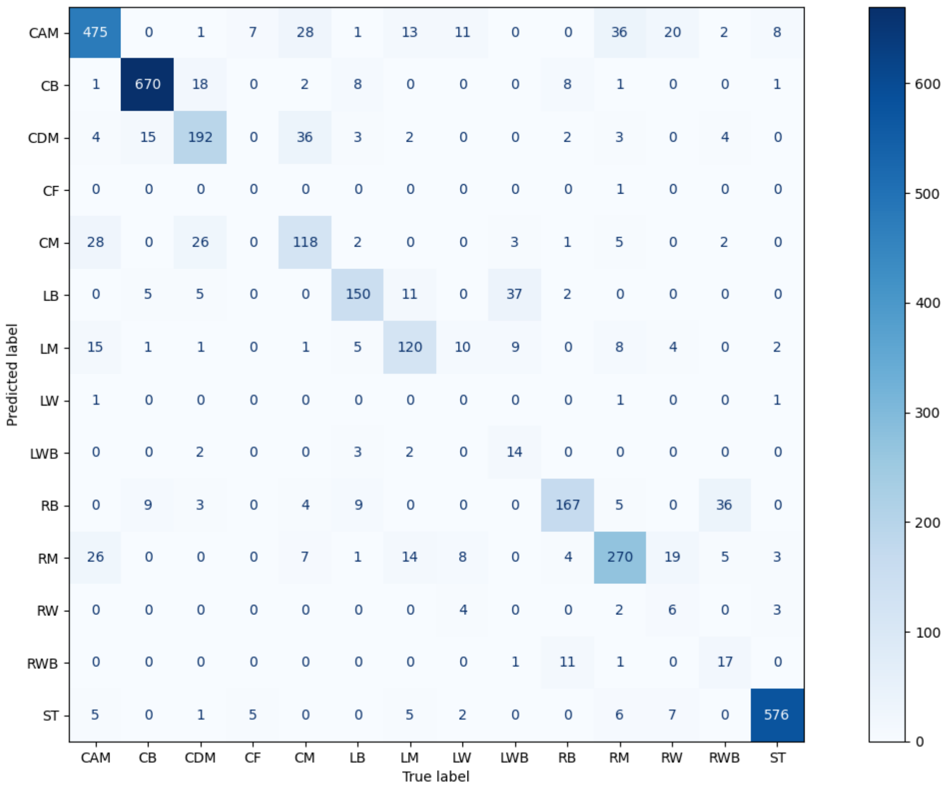
Naive Bayes is a fast technique in which the main assumption is the independence between each predictor of the training set, allowing to simplify the computation of $P(x_p|k)$ as seen in Section § 6.5. Assume that the distribution of $P(x_p|k)$ is multinomial. Then, Naive Bayes is applied using the *MultinomialNB* function, available from the Python *sklearn.naive_bayes* library (see *Function MultinomialNB*). This function can be applied because each value in the training set is typed as integer. No 10-Fold Cross Validation is required due to the absence of hyperparameters to tune. So, we simply build the standard multinomial naive bayes classifier from the training set, in which the *alpha* value (a.k.a. *equivalent sample size*) is set to 1.0. After the training phase, the Training Accuracy is very low and equal to 0.476, meaning that the classifier is not stable due to a random classification. Next, we make predictions on the fitted model, giving in input the test set and obtaining a Test Accuracy equal to 0.458. The model does not overfit due to a little difference between the Training Accuracy value and the Test Accuracy value, but the values are very unsatisfactory. Finally, the returned AUC value is set to 0.862 and we can suggest that the over-all class-specific performance is quite satisfactory. For what concerns the sensitivity values, it follows that CB, ST, CM and CDM positions are the best predicted, with a sensitivity value of 0.73, 0.64, 0.59 and 0.53. All the other positions have sensitivity values lower than 0.5 and they are badly predicted.
In order to check possible improvements, *GaussianNB* function is tried, available from the Python *sklearn.naive_bayes* library (see *Function GaussianNB*). Differently from *MultinomialNB* function, the assumption is that the distribution of $P(x_p|k)$ is Gaussian. No 10-Fold Cross Validation is required due to the absence of hyperparameters to tune. So, we simply build the standard Gaussian naive bayes classifier from the training set. After the training phase, the Training Accuracy is equal to 0.522, meaning that the classifier is not stable due to a random classification. Next, we make predictions on the fitted model, giving in input the test set and obtaining a Test Accuracy equal to 0.499. As in the multinomial naive bayes classifier, the model does not overfit. Nevertheless, the Training Accuracy value and the Test Accuracy values are not satisfactory and the model is not reliable in prediction. Finally, the returned AUC value is set to 0.882 and we can suggest that the over-all class-specific perfor-

mance is quite satisfactory. For what concerns the sensitivity values, it follows that CB, ST, CF, RM and RWB positions are the best predicted, with a sensitivity value of 0.82, 0.72, 0.62, 0.6 and 0.55. All the other positions have sensitivity values between 0.3 and 0.5, with the worst predicted RW with a sensitivity value equal to 0.12.
In conclusion, we can deduce that Naive Bayes technique is the worst Machine Learning technique seen until now according to the metrics values. These bad results may be caused by failure to meet the Naive Bayes assumption, because we have previously suggested that predictors are somehow correlated and therefore their independence cannot be achieved.
The code concerning the Naive Bayes technique is shown in Listing § B.14.

7.6 Support Vector Machine

SVM is the last Machine Learning technique we analyzed, in which the target is to define non-linear boundaries in a p-dimensional space for predicting the class of a new instance, given the training set observations. SVM technique is applied using the *SVC* function, available from the Python *sklearn.svm* library (see *Function SVC*). The standardized version of training set and test set is used because SVM technique is not scale invariant. We analyzed different version of SVM classifiers by implementing the three different kernel functions seen in Section § 6.6, which are *linear*, *polynomial* and *rbf*. The kernel function is a similarity function that transforms the standardized training set observations in order to define the boundaries. Different hyperparameters have been tried for each type of kernel function through 10-Fold Cross-Validation.
First, consider *linear* kernel function. In a first trial, we build the default SVM classifier from the standardized training set, in which the linear kernel function is applied together with the regularization parameter set to 1.0. After the training phase, the Training Accuracy is quite satisfactory and it is equal to 0.839. Next, we make predictions on the fitted model, giving in input the standardized test set and obtaining a Test Accuracy equal to 0.813. We can deduce that the default model does not overfit and that the Training Accuracy value and the Test Accuracy value are quite satisfactory. Nevertheless, we want to see if improvements can be done. So, 10-Fold Cross-Validation has been applied in order to look for the best hyperparameters to set. In this case, C is the only hyperparameter to check and it is the regularization tuning parameter that indicates how much to tolerate the margin errors. 0.1, 0.5 and 1.0 values are considered as C values. After having applied 10-Fold Cross-Validation, the best model found has C hyperparameter equal to 0.5, a lower value than the standard SVM

and so causing to reduce the bias and to increase the variance. The Training Accuracy is 0.836 and it is quite satisfactory. For what concerns the prediction phase, the Test Accuracy is set to 0.814, so slightly bigger than the Test Accuracy found with the default model. These results suggest that the model does not overfit and, in addition to it, the difference between the Training Accuracy and Test Accuracy is lower than the difference in the default model. So, the results obtained are generally good. Finally, the returned AUC value is set to 0.975, the best found until now, and we can suggest that the overall class-specific performance is almost perfect. For what concerns the sensitivity values, it follows that ST, CB, CAM, RB, LB, RM, CDM, LM and CM positions are the best predicted, with a sensitivity values of 0.97, 0.96, 0.86, 0.86, 0.82, 0.80, 0.77, 0.72, and 0.6, respectively. All the other positions are badly predicted, with sensitivity values lower than 0.26. Figure 7.8 shows the confusion matrix obtained in prediction phase. We can conclude that all the central defensive roles, all the midfielder positions and the striker position can be reliably predicted by the model.



	CAM	CB	CDM	CF	CM	LB	LM	LW	LWB	RB	RM	RW	RWB	ST
CAM	475	0	1	7	28	1	13	11	0	0	36	20	2	8
CB	1	670	18	0	2	8	0	0	0	8	1	0	0	1
CDM	4	15	192	0	36	3	2	0	0	2	3	0	4	0
CF	0	0	0	0	0	0	0	0	0	0	1	0	0	0
CM	28	0	26	0	118	2	0	0	3	1	5	0	2	0
LB	0	5	5	0	0	150	11	0	37	2	0	0	0	0
LM	15	1	1	0	1	5	120	10	9	0	8	4	0	2
LW	1	0	0	0	0	0	0	0	0	0	1	0	0	1
LWB	0	0	2	0	0	3	2	0	14	0	0	0	0	0
RB	0	9	3	0	4	9	0	0	0	167	5	0	36	0
RM	26	0	0	0	7	1	14	8	0	4	270	19	5	3
RW	0	0	0	0	0	0	0	4	0	0	2	6	0	3
RWB	0	0	0	0	0	0	0	0	1	11	1	0	17	0
ST	5	0	1	5	0	0	5	2	0	0	6	7	0	576

Figure 7.8: Confusion matrix in prediction phase for SVM with *linear* kernel.

Then, consider *polynomial* kernel function. In a first trial, we build the default SVM classifier from the standardized training set, in which the polynomial kernel function is applied, the regularization parameter is set to 1.0, the kernel coefficient *gamma* is set to 1.0, the independent term of kernel function *coef0* is set to 1 and finally the degree of the polynomial kernel function is set to 3. After the training phase, a perfect Training Accuracy equal to 1.0 is returned, so the model fits perfectly the standardized training set observations. Next, we make predictions on the fitted model, giving in input the standardized test set and obtaining a Test Accuracy equal to 0.729. Even though the Test Accuracy value is quite satisfactory, the model could overfit due to the high difference between the Training Accuracy value and the test Accuracy value. So, 10-Fold Cross-Validation has been applied in order to look for the best hyperparameters to set, which are:

- *C*, the regularization tuning parameter that indicates how much to tolerate the margin errors, considering 0.1, 0.5 and 1.0 values;

- *degree*, the non-negative integer degree of the polynomial kernel function, considering 1, 2 and 3 values.

After having applied 10-Fold Cross-Validation, the best model found has *C* hyperparameter equal to 0.5 and *degree* hyperparameter set to 1. From the best hyperparameters found, we can assume that the bias should reduce and the variance should increase due to a lower *C* value than the one in the standard model; moreover, the *degree* is lower than the one in the standard model, so the decision boundaries should be less flexible. Not too surprisingly, all the metrics values as Training Accuracy, Test Accuracy, AUC and Sensitivity are equal to those obtained with the application of the 10-Fold Cross-Validation in the *linear* kernel function. These results are equal because the *degree* is equal to 1, making the *linear* and *polynomial* kernel functions almost equal.
Finally, consider *rbf* kernel function. In a first trial, we build the default SVM classifier from the standardized training set, in which the rbf kernel function is applied, the regularization parameter is set to 1.0 and the kernel coefficient *gamma* is set to *1/(total number of attributes of the standardized training set * the standardized training set variance).* After the training phase, the Training Accuracy is quite satisfactory and it is equal to 0.835. Next, we make predictions on the fitted model, giving in input the standardized test set and obtaining a Test Accuracy equal to 0.773. We can deduce that the default model does not overfit and that the Training Accuracy value and the Test Accuracy value are quite satisfactory. Nevertheless, we

want to see if improvements can be done. So, 10-Fold Cross-Validation has been applied in order to look for the best hyperparameters to set, which are:

- *C*, the regularization tuning parameter that indicates how much to tolerate the margin errors, considering 0.1, 0.5 and 1.0 values;
- *gamma*, the positive kernel coefficient that indicates the influence of a standardized training set observation, considering 0.1, 0.01 and 0.001 values.

After having applied 10-Fold Cross-Validation, the best model found has *C* hyperparameter equal to 1 and *gamma* hyperparameter equal to 0.01. The Training Accuracy is 0.819 and it is quite satisfactory, even though its value is lower than the Training Accuracy found in the default model. For what concerns the prediction phase, the Test Accuracy is set to 0.782, so slightly bigger than the Test Accuracy found with the default model. These results suggest that the model does not overfit and, in addition to it, the difference between the Training Accuracy and Test Accuracy is lower than the difference in the default model. Finally, the returned AUC value is 0.97, and we can suggest that the overall class-specific performance is almost perfect. For what concerns the sensitivity values, it follows that ST, CB, CAM, RB, LB, CDM, RM, LM and CM positions are the best predicted, with a sensitivity values of 0.95, 0.94, 0.87, 0.86, 0.85, 0.75, 0.70, 0.66, and 0.54, respectively. All the other positions are badly predicted, with sensitivity values lower than 0.05. We can conclude that all the central defensive roles and the striker position can be reliably predicted by the model, while worse performances have been encountered in predicting all the midfielder positions with respect to the previous SVM classifiers.
In conclusion, SVM is the best Machine Learning technique seen from the metrics values point of view. The implementation of the three different kernel functions demonstrates how much the performances are similar in terms of results, with slightly better results for *linear* and *polynomial* kernel.
The code concerning the SVM technique is shown in Listing § B.15.

8
Relevel response variable classes

Data Mining and Machine Learning techniques have been applied in the previous chapters, demonstrating that some techniques such as Multinomial Logistic Regression, Lasso, and SVM, outperform alternatives in terms of metrics results. Nonetheless, one serious problem plagues even the best algorithms, namely, the lateral positions and low-frequency positions are badly predicted. This chapter faces this problem by relevelling the classes of the response variable, in particular reducing the number of levels with the unification of some classes. The reduction of the response variable classes is a data-driven process, in which the confusion matrices of the best techniques obtained after the prediction phase are considered. At the end, this modification is applied in each of the best techniques found and the metrics results will be analyzed.

8.1 Classes reduction process

The original number of classes for the response variable *Best_Position* is fourteen, specifying all the possible spatial position of a footballer in a pitch. Anyway, the number of levels is too high to guarantee satisfactory predictions for each player position. This fact is confirmed by the sensitivity values for lateral positions and low-frequency positions, discussed in Chapter § 5 and Chapter § 7. More in detail, this phenomenon occurs for positions with few observations, as they are often confused with more frequent classes with which they share some characteristics. According to the response variable distribution (see Figure 3.1), the

dataset is highly unbalanced. The consequence is that, as observed in the prediction phase of Data Mining and Machine Learning techniques, the simplest players' positions to detect are those corresponding to the most frequent observations, with bad prediction results for the least frequent players' positions.
Let Figure 5.2, Figure 5.14, and Figure 7.8 be the confusion matrices of Multinomial Logistic Regression, Lasso, and SVM with linear kernel, respectively. These methods return great results in Training Accuracy, Test Accuracy, and AUC metrics, with very similar values within each technique. However, all these metrics only take into account the overall average, effectively masking the negative effects of poorly predicted classes. Sensitivity metric comes to our aid in understanding the negative effects in prediction. Consider each column of the confusion matrices, which corresponds to a specific player position, and focus on false negatives. The most under-predicted classes for each position are reported below.

- **CB** is mostly predicted as CDM, followed by RB and LB. Anyway, the misclassification number is negligible if compared with the total number of true positives plus false negatives.

- **RB** is mostly predicted as RWB, followed by CB and RM. Anyway, the misclassification number is negligible if compared with the total number of true positives plus false negatives.

- **LB** is mostly predicted as LWB, followed by CB and LB. Anyway, the misclassification number is negligible if compared with the total number of true positives plus false negatives.

- **RWB** is mostly predicted as RB, followed by RM. Unfortunately, the misclassification number is significant if compared with the total number of true positives plus false negatives, and the sensitivity value is completely unsatisfactory.

- **LWB** is mostly predicted as LB, followed by LM. Unfortunately, the misclassification number is significant if compared with the total number of true positives plus false negatives, and the sensitivity value is completely unsatisfactory.

- **CDM** is mostly predicted as CM, followed by CB. Anyway, the misclassification number is negligible if compared with the total number of true positives plus false negatives.

- **CM** is mostly predicted as CDM, followed by CAM and RM. Even though the misclassification number is less than 50% if compared with the total number of true positives plus false negatives, we suggest that the sensitivity value can be improved.

- **RM** is mostly predicted as CAM, followed by LM. Anyway, the misclassification number is quite negligible if compared with the total number of true positives plus false negatives.

- **LM** is mostly predicted as RM, followed by CAM and LB. Anyway, the misclassification number is quite negligible if compared with the total number of true positives plus false negatives.

- **CAM** is mostly predicted as CM, followed by RM and LM. Anyway, the misclassification number is negligible if compared with the total number of true positives plus false negatives.

- **CF** is mostly predicted as CAM, followed by ST. Unfortunately, the misclassification number is significant if compared with the total number of true positives plus false negatives, and the sensitivity value is completely unsatisfactory.

- **RW** is mostly predicted as RM, followed by CAM and ST. Unfortunately, the misclassification number is significant if compared with the total number of true positives plus false negatives, and the sensitivity value is completely unsatisfactory.

- **LW** is mostly predicted as LM, followed by RM and CAM. Unfortunately, the misclassification number is significant if compared with the total number of true positives plus false negatives, and the sensitivity value is completely unsatisfactory.

- **ST** is mostly predicted as CAM. Unfortunately, the misclassification number is significant if compared with the total number of true positives plus false negatives, and the sensitivity value is completely unsatisfactory.

Predicting classes as RWB, LWB, CF, RW, and LW implies to obtain poor prediction results due to few observations on the dataset for those players' positions. So, the metrics values could be improved unifying some classes which share common features according to the data obtained in prediction phase from confusion matrices. This process allows to augment the observations for a specific new class, ideally improving the single-class prediction results and the overall performance of models. Furthermore, nowadays football is a dynamic game in which a versatile player is capable to play multiple roles during the match, so the unification process is not unrealistic. The new levels of response variable *Best_Position* are listed below, followed by a justification for those unified classes.

- **CB**, as in the original dataset.

- **RB-RWB**, that unifies RB and RWB defensive positions, placed at the right side of the pitch and close geographically. The unification is applied because, when the prediction is wrong, RB is predicted as RWB in most cases and vice versa.

- **LB-LWB**, that unifies LB and LWB defensive positions, placed at the left side of the pitch and close geographically. The unification is applied because, when the prediction is wrong, LB is predicted as LWB in most cases and vice versa.

- **CDM-CM**, that unifies CDM and CM midfielder positions, placed at the middle of the pitch and close geographically. The unification is applied for improving CM prediction performances, which is predicted as CDM when the prediction is wrong and vice versa.

- **RM-RW**, that unifies RM midfielder position and RW forward position, placed at the right side of the pitch and each one with different tasks. The unification is applied to improve the poor prediction performance of RW, due to a high presence of ST forward role. Moreover, when the prediction is wrong, RW is predicted as RM in most cases.

- **LM-LW**, that unifies LM midfielder position and LW forward position, placed at the left side of the pitch and each one with different tasks. The unification is applied to improve the poor prediction performance of LW, due to a high presence of ST forward role. Moreover, when the prediction is wrong, LW is predicted as LM in most cases.

- **CAM-CF**, that unifies CAM midfielder position and CF forward position, placed at the middle of the pitch and close geographically. The unification is applied to improve the poor prediction performance of CF, due to a high presence of ST forward role. Moreover, when the prediction is wrong, CF is predicted as CAM in most cases.

- **ST**, as in the original dataset.

The number of classes switches from fourteen to eight. Unified positions, marked with "-", can be read as the capacity of a football player to employ different roles. Figure 8.1 shows the new distribution of the response variable *Best_Position*. We can see that there are levels as CB, ST and CAM-CF which prevail over the other levels. The least frequent level is LM-LW and it represents almost the 6% of the total observations, so a relevant number. Moreover, the number of observations for RB-RWB and LW-LWB levels is very similar. From the reduction of the response variable levels, we can suggest that each class can be predicted satisfactorily due to an acceptable number of observations for each position, with a little improvement on metrics values.

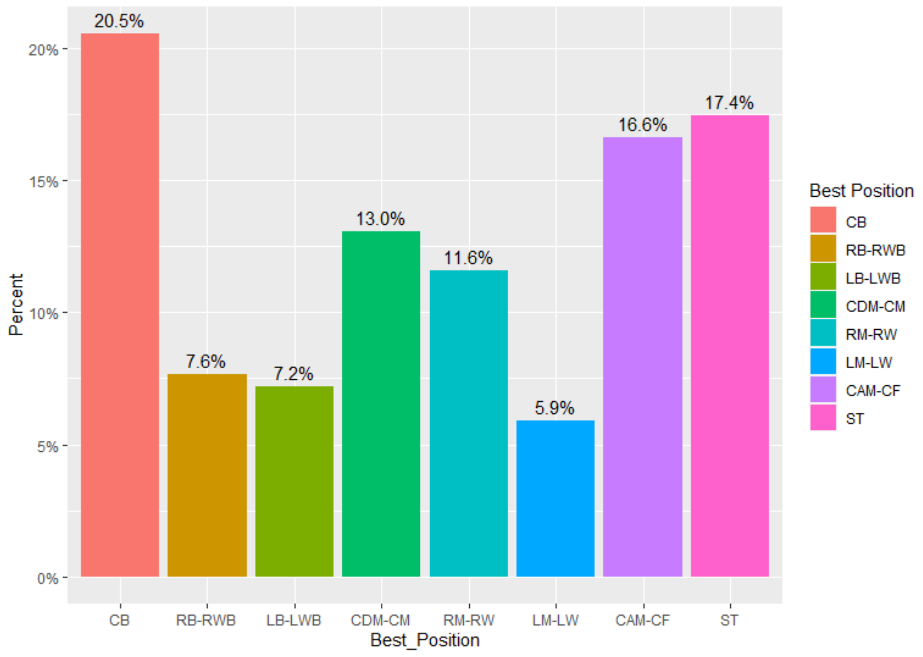


Figure 8.1: Response variable distribution after relevelling.

The code concerning the relevelling of the response variable levels is the same shown in Listing § A.1, except for row 51 which is substituted with the code below.

```
levels(new_data$Best_Position) <- c("CB", "RB-RWB", "LB-LWB", "RB-RWB", "LB-
    LWB", "CDM-CM", "CDM-CM", "RM-RW", "LM-LW", "CAM-CF", "CAM-CF", "RM-RW",
     "LM-LW", "ST")
```

8.2 Multinomial Logistic Regression

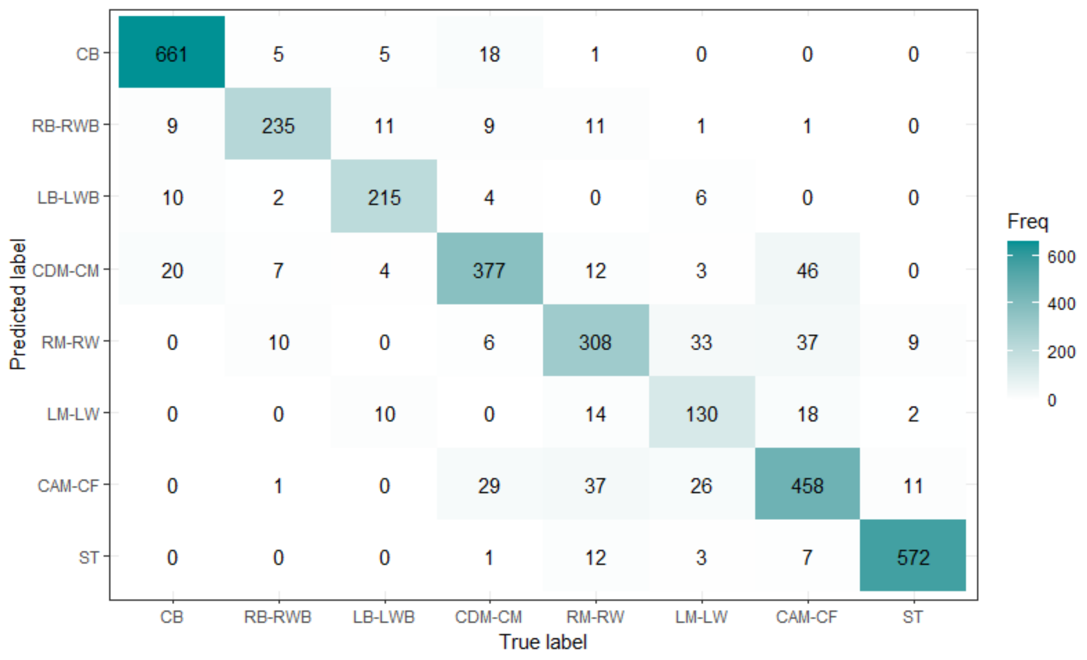
Multinomial Logistic Regression is the first technique applied to the new relevelling of the response variable. As seen in Section § 5.2, the best model found includes all the variables plus ten interactions, which are all significant. Hence, no modifications have been applied before training the model. After the training phase, it has been confirmed that each variable and interaction is significant. Moreover, the Training Accuracy value is 0.878 with a 95% confidence interval [0.8724, 0.8835], the AIC value is 9935.718 and the BIC value is 12935.65. From this results, we can deduce the model fits better than the original model due to the lowering of AIC and BIC values. Another point to analyze is the variable importance, given by the sum of the absolute values of the coefficients estimates for each level. The variable importance values are reported in Table 8.1.

Table 8.1: Variable importance for Multinomial Logistic Regression after relevelling response variable levels. AWR = Attacking_Work_Rate, DWR = Defensive_Work_Rate, BTL = Body_TypeLean, BTN = Body_TypeNormal, BTS = Body_TypeStocky, SP:BC = Short_Passing:Ball_Control, SP:LP = Short_Passing:Long_Passing, A:SS = Acceleration:Sprint_Speed, C:FKA = Curve:Free_Kick_Accuracy.

Height_cm	Weight_kg	Preferred_FootRight	Crossing
0.599	0.139	14.573	1.592
Finishing	Heading_Accuracy	Short_Passing	Volleys
0.865	2.068	1.663	0.297
Dribbling	Curve	Free_Kick_Accuracy	Long_Passing
0.643	0.206	0.174	2.048
Ball_Control	Acceleration	Sprint_Speed	Agility
1.174	2.349	2.276	0.928
Reactions	Balance	Shot_Power	Jumping
0.197	0.112	0.366	0.740
Stamina	Strength	Long_Shots	Aggression
0.971	1.542	0.520	1.565
Interceptions	Positioning	Vision	Penalties
3.846	0.818	1.411	0.201
Composure	Weak_Foot2	Weak_Foot3	Weak_Foot4
0.215	11.567	11.779	14.402
Weak_Foot5	AWRMedium	AWRHigh	DWRMedium
15.152	4.787	5.844	1.637
DWRHigh	BTL (170-185)	BTL (185+)	BTN (170-)
22.566	3.331	3.727	23.368
BTN (170-185)	BTN (185+)	BTS (170-)	BTS (170-185)
10.153	5.135	8.753	3.574
BTS (185+)	Body_TypeUnique	SP:BC	Aggression:Interceptions
45.756	7.395	0.018	0.011
SP:LP	Finishing:Long_Shots	A:SS	Dribbling:Ball_Control
0.018	0.009	0.027	0.006
Curve:FKA	Long_Passing:Vision	Dribbling:Agility	Ball_Control:Agility
0.001	0.014	0.017	0.019

The variable importance obtained is lower than the original variable importance for each variable, except for some interactions which have higher estimates. In general, the most important predictors are *Body_ TypeStocky (185+)*, *Body_ TypeNormal (170-)*, *Defensive_ Work_RateHigh*, *Defensive_ Work_RateMedium*, *Weak_Foot5*, *Preferred_FootRight*, *Weak_Foot4*, *Weak_Foot3*, *Weak_Foot2*, and *Body_ TypeNormal (170-185)*.

For what concerns the prediction phase, the Test Accuracy value is equal to 0.868 with a 95% confidence interval [0.856, 0.879], a very satisfactory value close to the Training Accuracy value which confirms the model generalizes well. Moreover, the AUC value is 0.918 and almost close to 1, which means perfect classification. As expected, sensitivity values have been visibly improved than the ones obtained in the original model. Indeed, the "worst"predicted position is LM-LW, with a sensitivity value of 0.643. ST is the best predicted position together with CB, with sensitivity values equal to 0.9622 and 0.9443, respectively. For what concerns RB-RWB, LB-LWB, CDM-CM, CAM-CF, and RM-RW positions, the sensitivity values are equal to 0.904, 0.878, 0.849, 0.8078, and 0.78, respectively. Figure 8.2 shows the confusion matrix obtained in prediction phase.



Predicted label \ True label	CB	RB-RWB	LB-LWB	CDM-CM	RM-RW	LM-LW	CAM-CF	ST
CB	661	5	5	18	1	0	0	0
RB-RWB	9	235	11	9	11	1	1	0
LB-LWB	10	2	215	4	0	6	0	0
CDM-CM	20	7	4	377	12	3	46	0
RM-RW	0	10	0	6	308	33	37	9
LM-LW	0	0	10	0	14	130	18	2
CAM-CF	0	1	0	29	37	26	458	11
ST	0	0	0	1	12	3	7	572

Figure 8.2: Confusion matrix in prediction phase for Multinomial Logistic Regression after relevelling.

8.3 Lasso

Lasso is the best regularization method in which coefficients estimates are shrunk towards zero and variable selection is applied. As seen in Subsection § 5.5.2, the model includes the same covariates of the best Multinomial Logistic Regression model. The training phase procedure is the same. First, it has been checked the Lasso behaviour in which the higher the $ln(\lambda)$ the more variables are set to 0. Then, 10-Fold Cross-Validation technique has been applied to find the best λ value, according to the minimum misclassification error.

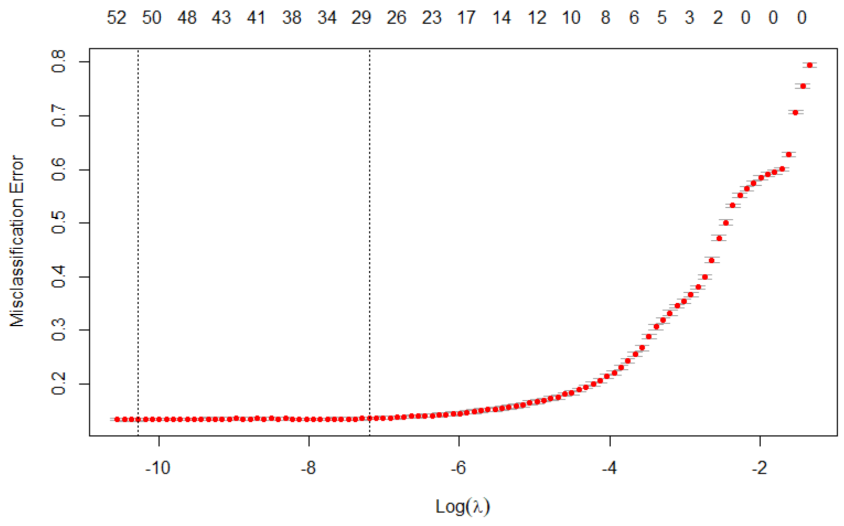


Figure 8.3: Misclassification error varying λ value during the 10-Fold cross validation with Lasso after response variable relevelling. The leftmost vertical dashed line is the λ that corresponds to the minimum misclassification error, and the rightmost vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error.

Figure 8.3 shows that the higher the $ln(\lambda)$ the higher the misclassification error. Moreover, misclassification error increases rapidly from $ln(\lambda) = -4$. Then, there are two different λ values. The leftmost vertical dashed line is the λ that corresponds to the minimum misclassification error, and the rightmost vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error. The minimum misclassification error corresponds to 0.134 and the minimum misclassification error plus 1 standard error is equal to 0.136. The difference between the two latter error values is almost negligible, so the λ that corresponds to the minimum misclassification error plus 1 standard error has been preferred due to the highest variable selection, in which the variables maintained in the model are 29. The best λ has a value of 7.484e-04. Moreover, Figure 8.4 shows that the maximum explained deviance obtained from the best λ is 0.818, and so higher than the maximum explained deviance obtained by the original model.

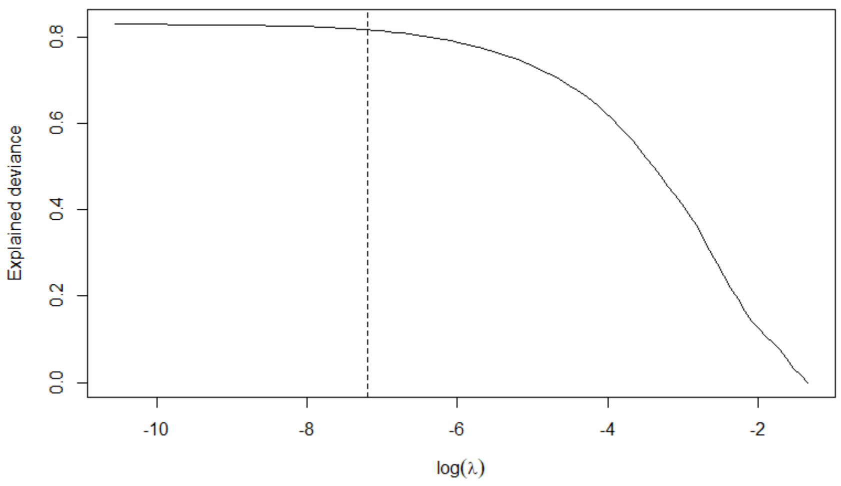


Figure 8.4: Explained deviance according to λ value in Lasso after response variable relevelling. The vertical dashed line is the λ that corresponds to the minimum misclassification error plus 1 standard error.

For what concerns the metrics values of the training phase, the Training Accuracy value is 0.869 with a 95% confidence interval [0.863, 0.875], the AIC value is -44330.7 and the BIC value is -43917.63. The results are very satisfactory, but slightly worse than the original model. Finally, the variable importance computed for each response variable level demonstrates that the coefficients estimates are slightly lower.
For what concerns the metrics values of the prediction phase, the Test Accuracy value is very close to the Training Accuracy value and it is equal to 0.866 with a 95% confidence interval [0.855, 0.878], guaranteeing the model generalizes well. Moreover, the AUC value is 0.981 and almost close to 1, which means perfect classification. As expected, sensitivity values have been visibly improved than the ones obtained in the original model. As seen in Section § 8.2, ST and CB are the best predicted positions, with sensitivity values of 0.963 and 0.947, respectively. Moreover, LM-LW remains the most misclassified position, even though its sensitivity value is above the 50% and equal to 0.619. For what concerns RB-RWB, LB-LWB, CDM-CM, CAM-CF, and RM-RW positions, the sensitivity values are equal to 0.9, 0.894, 0.863, 0.801, and 0.765, respectively. Figure 8.5 shows the confusion matrix obtained in

prediction phase.

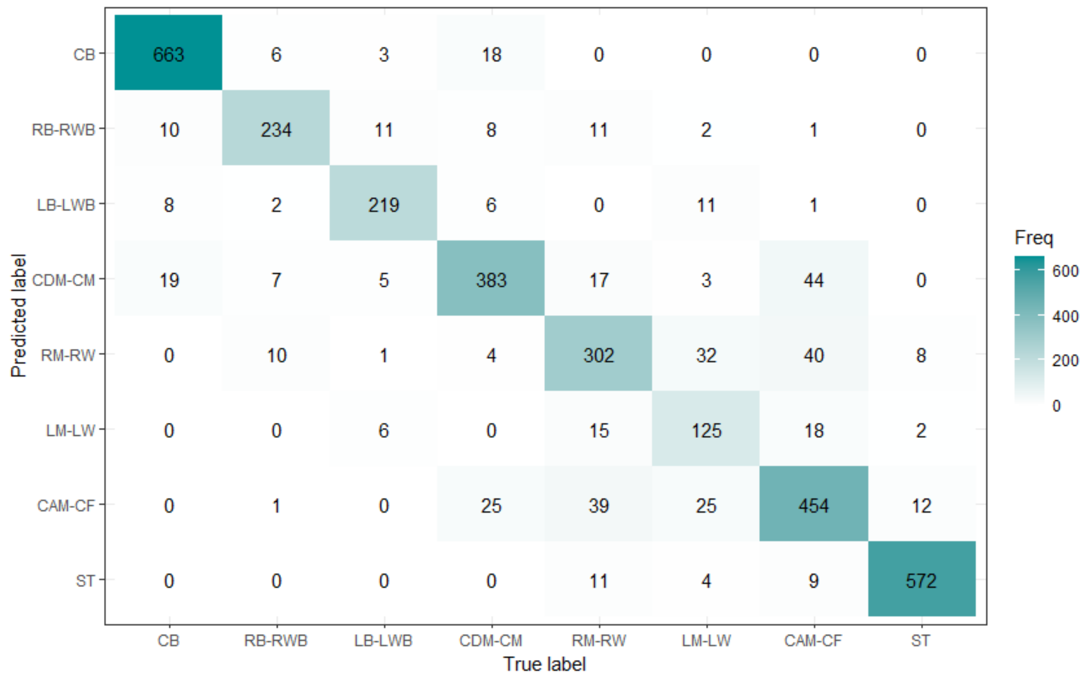


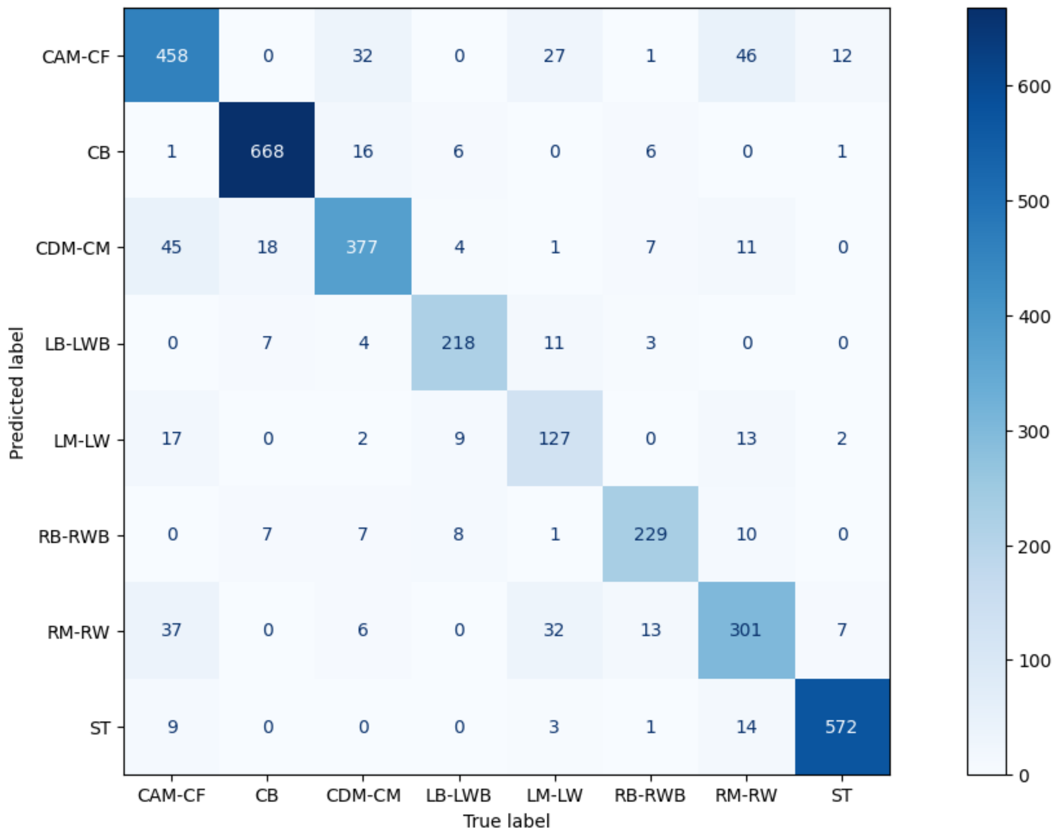
Figure 8.5: Confusion matrix in prediction phase for Lasso after relevelling.

The code concerning Lasso applied after the relevelling of the response variable is the same shown in Listing § A.7, except for rows 29, 32 and 36, in which the generic function *c* is substituted with the code below.

```
c ( ”CB” , ”RB-RWB” , ”LB-LWB” , ”CDM-CM” , ”RM-RW” , ”LM-LW” , ”CAM-CF” , ”ST” )
```

8.4 Support Vector Machine

SVM is the last technique in which the relevelling of the response variable has been tested. As seen in Section § 7.6, the model includes the same covariates of the best Multinomial Logistic Regression model. In addition to it, the dataset is standardized in order to avoid negative scale effects. The training phase procedure is the same, using the *linear* kernel function, the best one found. First, a default SVM has been built from the standardized training set, in which the linear kernel function is applied together with the regularization parameter set to 1.0. After the training phase, the Training Accuracy is very satisfactory and it is equal to 0.885. Next, we make predictions on the fitted model, giving in input the standardized test set and obtaining a Test Accuracy equal to 0.866. We can deduce that the default model does not overfit and that the Training Accuracy value and the Test Accuracy value are very satisfactory. Anyway, 10-Fold Cross-Validation has been applied in order to look for the best regularization hyperparameter *C* within the values 0.1, 0.5 and 1.0. After having applied 10-Fold Cross-Validation, the best model found has exactly the same C hyperparameter applied for the default model, that is equal to 1.0. Finally, the returned AUC value is set to 0.984, suggesting that the overall class-specific performance is almost perfect.



Predicted label \ True label	CAM-CF	CB	CDM-CM	LB-LWB	LM-LW	RB-RWB	RM-RW	ST
CAM-CF	458	0	32	0	27	1	46	12
CB	1	668	16	6	0	6	0	1
CDM-CM	45	18	377	4	1	7	11	0
LB-LWB	0	7	4	218	11	3	0	0
LM-LW	17	0	2	9	127	0	13	2
RB-RWB	0	7	7	8	1	229	10	0
RM-RW	37	0	6	0	32	13	301	7
ST	9	0	0	0	3	1	14	572

Figure 8.6: Confusion matrix in prediction phase for SVM after relevelling

As expected, sensitivity values have been visibly improved than the ones obtained in the original model. As seen in Section § 8.2 and Section § 8.3, ST and CB are the best predicted positions, with sensitivity values of 0.96 and 0.95, respectively. Moreover, LM-LW remains the most misclassified position, even though its sensitivity value is above the 50% and equal to 0.63. For what concerns LB-LWB, RB-RWB, CDM-CM, CAM-CF, and RM-RW, the

sensitivity values are equal to 0.89, 0.88, 0.85, 0.81, and 0.76, respectively. Figure 8.6 shows the confusion matrix obtained in prediction phase.
The code concerning SVM applied after the relevelling of the response variable is the same shown in Listing § B.15 (consider only SVM with *linear* kernel, rows 6-16). Anyway, it is necessary to substitute rows 6 and 12 of Listing § B.7 with

```
training_set["Best_Position"] = training_set["Best_Position"].cat.set_
    categories(['CB', 'RB-RWB', 'LB-LWB', 'CDM-CM', 'RM-RW', 'LM-LW', 'CAM-
    CF', 'ST'], ordered=True)

test_set["Best_Position"] = test_set["Best_Position"].cat.set_categories(['
    CB', 'RB-RWB', 'LB-LWB', 'CDM-CM', 'RM-RW', 'LM-LW', 'CAM-CF', 'ST'],
    ordered=True)
```

to reproduce the same results.

9 Conclusions

This chapter concludes the thesis with the aim of summarizing all the results of the applied Data Mining techniques and Machine Learning techniques. The final summary focuses on comparing each model according to the metrics used in both the training phase and the prediction phase. Moreover, the execution time in the training phase is considered. In particular, the time spent for both cross-validation and model fitting with the best hyperparameters (if any) has been computed.

Table 9.1: Metrics values results for every technique. Character "/" means that the value cannot be computed for the specified method.

	Training Accuracy	**AIC**	**BIC**	**Test Accuracy**	**AUC**
Multinomial Logistic Regression	0.828	14441.09	20012.24	0.818	0.963
Backward model selection	0.824	14265.7	17686.58	0.817	0.965
LDA	0.742	/	/	0.718	0.88

Ridge Regression	0.757	-38773.36	-38352.8	0.748	0.93
Lasso	0.821	-47461.61	-47041.05	0.811	0.962
Decision Tree	0.846	/	/	0.655	0.816
(Pruned) Decision Tree	0.731	/	/	0.675	0.907
Random Forest	1.0	/	/	0.753	0.951
KNN	1.0	/	/	0.664	0.928
(Gaussian) Naive Bayes	0.522	/	/	0.498	0.882
(Linear) SVM	0.836	/	/	0.814	0.975

Table 9.1 reports the summary of the Data Mining and Machine Learning techniques results in terms of metrics values. For what concerns Data Mining techniques, the Multinomial Logistic Regression (together with the model obtained after backward variable selection) and Lasso are the preferable solutions. Both the models generalize well and the accuracy values, together with the AUC value, are very satisfactory. Differently from the Multinomial Logistic Regression, Lasso performs variable selection and it has the lowest AIC and BIC values, suggesting that the model fits better than the others. Although they do not overfit, LDA and Ridge Regression have less satisfactory results because LDA does not respect the assumptions and Ridge Regression excessively shrinks the coefficients without doing variable selection. For what concerns Machine Learning techniques, the SVM with linear kernel function (or polynomial, the results are equal) is the best model in terms of results. Moreover, the accuracy and AUC values are very similar to those obtained by Multinomial Logistic Regression and Lasso. The pruned Decision Tree is the second best model because it improves in generalization with respect to the Decision Tree without pruning, but the accuracy values are not so high. Moreover, Random Forest and KNN perfectly fit the training set instances, but their Test Accuracy values are too far from the Training Accuracy values and the models overfit. Finally, Naive Bayes is the worst model because it predicts the right players' position

the 50% of times, in other words, randomly.

Table 9.2: Training times for every technique. Character "/" means that the time is not computed because Cross-Validation has not been applied. Character * means that the value refers to the time spent for fitting all the Decision Trees pruned.

	Cross-Validation time (in seconds)	**Training time (in seconds)**
Multinomial Logistic Regression	/	101.72
Backward model selection	/	18557.5
LDA	/	0.67
Ridge Regression	53.97	5.37
Lasso	184.38	20.31
Decision Tree	215.48	0.45
(Pruned) Decision Tree	517.68 *	0.51
Random Forest	197575.94	5.34
KNN	1481.99	0.02
(Gaussian) Naive Bayes	/	0.04
(Linear) SVM	488.71	13.22

Table 9.2 reports the summary of the Data Mining and Machine Learning techniques results in terms of training execution time. For statistical reasons, Cross-Validation time has been computed for those methods which required to find the best hyperparameters, and this time can vary according to the possible combinations of hyperparameters to try. So, the training time is the one used to choose among the methods. For what concerns Data Mining techniques, LDA is the fastest method due to instant computation of prior probabilities and density functions, followed by Ridge Regression and Lasso. Multinomial Logistic Regression is fitted between one and two minutes, so the convergence time is very satisfactory. Backward model selection takes some hours due to the large number of predictors and the process of removing each time the worst variable basing on AIC metric, thus becoming unappealing with respect to alternatives. For what concerns Machine Learning techniques, all the methods require few seconds to be applied. More in detail, the fastest algorithms are KNN and Naive Bayes, followed by Decision Tree, Decision Tree pruned, Random Forest and finally SVM.

Only few algorithms have demonstrated to perform well in the context of prediction of the football players' positions, both in terms of results and execution time. Multinomial Logistic Regression, Lasso and SVM are the best algorithms, with a Test Accuracy between 0.81 and 0.82 and AUC values between 0.96 and 0.97. Moreover, the algorithms are not computationally expensive. The backward model selection has excellent results, similar to those obtained by the algorithm mentioned before, but at the price of substantial computational time. Finally, the smallest execution times are KNN, Naive Bayes and LDA. However, it has been demonstrated that results from these approaches are not satisfactory for predicting the position of a new player instance. While the players' central positions are well predicted because they have a high number of observations, no satisfactory predictions are obtained at the players' lateral positions. For this reason, Multinomial Logistic Regression, Lasso and SVM have been considered as reference instrument for classification in the additional analysis with a reduced number of classes of the response variable.

Table 9.3: Metrics values results for the best techniques after relevelling of the response variable. Character "/" means that the value cannot be computed for the specified method.

	Training Accuracy	AIC	BIC	Test Accuracy	AUC
Multinomial Logistic Regression	0.878	9935.78	12935.65	0.868	0.982
Lasso	0.869	-44330.7	-43917.63	0.866	0.981
(Linear) SVM	0.885	/	/	0.866	0.984

Table 9.3 reports the summary of the results of best techniques found after reducing the response variable classes, in terms of metrics values. There are improvements for each technique, due to a better prediction for every player's position. Moreover, each model generalizes well due to a tiny difference between the Training Accuracy value and the Test Accuracy value, suggesting that no one model overfits. Each result is very similar, even though SVM results to slightly be the best model in terms of Training Accuracy value and AUC if we wanted to find a winning model.

Table 9.4: Training times for the best techniques after relevelling of the response variable. Character "/" means that the time is not computed because Cross-Validation has not been applied.

	Cross-Validation time (in seconds)	Training time (in seconds)
Multinomial Logistic Regression	/	35.32
Lasso	103.74	12.24
(Linear) SVM	391.49	12.87

Table 9.4 reports the summary of the results of best techniques found after reducing the response variable classes, in terms of training execution time. Let the training time be the most reliable and useful information about the execution time, as in the previous summary. The training time has drastically dropped for Multinomial Logistic Regression. Both the Cross-Validation time and training time is almost halved in Lasso, which results to be the fastest algorithm. The execution time of SVM with linear kernel is almost unchanged, and it remains very small.
In conclusion, we can assume that there could be some limits in predicting the best player's position when the number of classes is quite high. Indeed, the overall metrics suggest the best models fit the data and predict new instances satisfactorily, but these results hide the negative effects of those positions that are wrongly predicted in more than 50% of cases. So, the analysis conducted in reducing the number of response variable classes shows how the class-specific prediction is improved through the unification process; consequently, even the overall metrics are improved. At the end, it is confirmed that ST and CB are the best predicted positions in both the scenarios and the only ones which have not been involved in the unification process.
The work of this thesis can be extended in several ways. One example is the use of unsupervised learning algorithms, which aim at discovering specific patterns from the observations given in input. Unsupervised learning algorithms do not provide target values associated to each observation during the training process. Clustering, feature extraction and dimensionality reduction algorithms are possible solutions. Furthermore, synthetic data can be explored as alternative implementation for improving prediction. Unlike the reduction process of the response variable classes, the suggested solution allows to artificially create data according to specific conditions related to the statistical properties of the original dataset. This process allows to obtain larger balanced dataset for the training process, so that the multi-

nomial distribution can be applied to response classes of equal dimension. Synthetic data are useful when there are too few real-world observations for a specific class and therefore additional observations for that class can be useful to improve model performance.

A

R code

This chapter includes the *R* (R Core Team, 2022) code produced during the writing of this thesis, in order to ensure the replicability of the obtained results.

A.1 Preliminary data analysis of FIFA dataset

This code allows to pre-process the dataset located in the desired *path* before starting its graphical evaluation.

```r
data <- read.csv("path/FIFA22_Dataset.csv", sep=";", encoding="UTF-8")

data <- data[!duplicated(data), ]
sum(is.na(data))

data$Age <- as.integer(data$Age)
data$Height_cm <- as.integer(data$Height_cm)
data$Weight_kg <- as.integer(data$Weight_kg)
data$Crossing <- as.integer(data$Crossing)
data$Finishing <- as.integer(data$Finishing)
data$Heading_Accuracy <- as.integer(data$Heading_Accuracy)
data$Short_Passing <- as.integer(data$Short_Passing)
data$Volleys <- as.integer(data$Volleys)
data$Dribbling <- as.integer(data$Dribbling)
data$Curve <- as.integer(data$Curve)
data$Free_Kick_Accuracy <- as.integer(data$Free_Kick_Accuracy)
```

```r
data$Long_Passing <- as.integer(data$Long_Passing)
data$Ball_Control <- as.integer(data$Ball_Control)
data$Acceleration <- as.integer(data$Acceleration)
data$Sprint_Speed <- as.integer(data$Sprint_Speed)
data$Agility <- as.integer(data$Agility)
data$Reactions <- as.integer(data$Reactions)
data$Balance <- as.integer(data$Balance)
data$Shot_Power <- as.integer(data$Shot_Power)
data$Jumping <- as.integer(data$Jumping)
data$Stamina <- as.integer(data$Stamina)
data$Strength <- as.integer(data$Strength)
data$Long_Shots <- as.integer(data$Long_Shots)
data$Aggression <- as.integer(data$Aggression)
data$Interceptions <- as.integer(data$Interceptions)
data$Positioning <- as.integer(data$Positioning)
data$Vision <- as.integer(data$Vision)
data$Penalties <- as.integer(data$Penalties)
data$Composure <- as.integer(data$Composure)
data$Defensive_Awareness <- as.integer(data$Defensive_Awareness)
data$Standing_Tackle <- as.integer(data$Standing_Tackle)
data$Sliding_Tackle <- as.integer(data$Sliding_Tackle)
data$GK_Diving <- as.integer(data$GK_Diving)
data$GK_Handling <- as.integer(data$GK_Handling)
data$GK_Kicking <- as.integer(data$GK_Kicking)
data$GK_Positioning <- as.integer(data$GK_Positioning)
data$GK_Reflexes <- as.integer(data$GK_Reflexes)
data$Pace_Diving <- as.integer(data$Pace_Diving)
data$Shooting_Handling <- as.integer(data$Shooting_Handling)
data$Passing_Kicking <- as.integer(data$Passing_Kicking)
data$Dribbling_Reflexes <- as.integer(data$Dribbling_Reflexes)
data$Defending_Pace <- as.integer(data$Defending_Pace)
data$Physical_Positioning <- as.integer(data$Physical_Positioning)

data$Preferred_Foot <- as.factor(data$Preferred_Foot)
data$Best_Position <- factor(data$Best_Position, levels = c("CB", "RB", "LB"
    , "RWB", "LWB", "CDM", "CM", "RM", "LM", "CAM", "CF", "RW", "LW", "ST",
    "GK"))
as.ordered(data$Best_Position)
data$Weak_Foot <- as.factor(data$Weak_Foot)
data$Attacking_Work_Rate <- factor(data$Attacking_Work_Rate, levels = c("Low
    ", "Medium", "High"))
as.ordered(data$Attacking_Work_Rate)
data$Defensive_Work_Rate <- factor(data$Defensive_Work_Rate, levels = c("Low
    ", "Medium", "High"))
```

```
as.ordered(data$Defensive_Work_Rate)
data$Body_Type <- as.factor(data$Body_Type)

new_data <- subset(data, select = -c(Name, Age, GK_Diving, GK_Handling, GK_
     Kicking, GK_Positioning, GK_Reflexes, Pace_Diving, Shooting_Handling,
     Passing_Kicking, Dribbling_Reflexes, Defending_Pace, Physical_
     Positioning))
new_data <- new_data[!(new_data$Best_Position=="GK"),]
new_data$Best_Position <- droplevels(new_data$Best_Position)
```

Listing A.1: Pre-processing steps

A.2 Data Mining Results

This code allows to split the original dataset into training set and test set. Remember to set *set.seed(100)* to reproduce the same two sets in order to obtain the same results. Finally, both the training set and the test set are saved in a *.RData* file, in order to recover them for future purposes. This file is saved in a specific *path*.

```
set.seed(100)

training_set <- data[FALSE, ] # Only columns names and no one row
test_set <- data[FALSE, ] # Only columns names and no one row

for (i in 1:nlevels(data$Best_Position)){
  temp_data <- data[(data$Best_Position==names(table(data$Best_Position)[i])
      ), ]
  sample <- sample(nrow(temp_data), 0.8*nrow(temp_data), replace=FALSE)

  temp_training_set <- temp_data[sample, ]
  temp_test_set <- temp_data[-sample, ]

  training_set <- rbind(training_set, temp_training_set)
  test_set <- rbind(test_set, temp_test_set)

  rm(temp_data)
  rm(sample)
  rm(temp_training_set)
  rm(temp_test_set)
}

```

```r
save(training_set, test_set, file = "path/dataset_split.RData")
```

Listing A.2: Split the dataset (80/20)

This code concerns the fit of the Multinomial Logistic Regression models from the training set through the *multinom* function. The *maxit* parameter is set to 10000 in order to increase the maximum number of iterations. The goodness of the models has been checked through the *anova* function. Then, *varImp* function is applied to print the overall contribute of every variable inside the model. Finally, it is provided the code for computing the metrics for evaluating the final model.

```r
library(caret)
library(pROC)
library(nnet)

starting_model <- multinom(Best_Position ~ ., data=training_set, model=TRUE,
    maxit=10000)
updated_model <- multinom(Best_Position ~ . + Acceleration:Sprint_Speed +
    Ball_Control:Dribbling + Free_Kick_Accuracy:Curve + Vision:Long_Passing
    + Agility:Dribbling + Agility:Ball_Control, data=training_set, model=
    TRUE, maxit=10000)
final_model <- multinom(Best_Position ~ . + Short_Passing:Ball_Control +
    Aggression:Interceptions + Short_Passing:Long_Passing + Finishing:Long_
    Shots + Acceleration:Sprint_Speed + Dribbling:Ball_Control + Free_Kick_
    Accuracy:Curve + Vision:Long_Passing + Agility:Dribbling + Agility:Ball_
    Control, data=training_set, model=TRUE, maxit=10000)

anova(starting_model, updated_model, test="Chisq")
anova(updated_model, final_model, test="Chisq")

# Print the importance of the variables
varImp(final_model, scale=FALSE)

train_bp_predicted <- predict(final_model, newdata=training_set, "class")
train_cf <- confusionMatrix(train_bp_predicted, training_set$Best_Position,
    mode="everything")

test_bp_predicted <- predict(final_model, newdata=test_set, "class")
test_cf <- confusionMatrix(test_bp_predicted, test_set$Best_Position, mode="
    everything")

roc <- multiclass.roc(test_set$Best_Position, predict(final_model, newdata=
    test_set, "prob"))

```

```
# Metrics values for training phase
train_cf$overall[1] # Training Accuracy
AIC(model) # AIC
BIC(model) # BIC

# Metrics values for prediction phase
test_cf$overall[1] # Test Accuracy
roc$auc # AUC
```

Listing A.3: Multinomial Logistic Regression code

This code allows to apply automatic model selection with the backward strategy. Both the AIC metric and the BIC metric are considered. The code for computing the metrics useful to evaluate the selected model with backward is provided.

```
library(caret)
library(pROC)

backward_model_AIC <- step(model_best, direction="backward")
backward_model_BIC <- step(model_best, direction="backward", k=log(dim(
    training_set)[1]))

# Print the importance of the variables
varImp(backward_model_AIC, scale=FALSE)

train_bp_predicted <- predict(backward_model_AIC, newdata=training_set, "
    class")
train_cf <- confusionMatrix(train_bp_predicted, training_set$Best_Position,
    mode="everything")

test_bp_predicted <- predict(backward_model_AIC, newdata=test_set, "class")
test_cf <- confusionMatrix(test_bp_predicted, test_set$Best_Position, mode="
    everything")

roc <- multiclass.roc(test_set$Best_Position, predict(backward_model_AIC,
    newdata=test_set, "prob"))

# Metrics values for training phase
train_cf$overall[1] # Training Accuracy
AIC(model) # AIC
BIC(model) # BIC

# Metrics values for prediction phase
test_cf$overall[1] # Test Accuracy
```

```
roc$auc # AUC
```

Listing A.4: Backward selection with AIC metric and BIC metric

This code allows to check the normality of the quantitative variables, to apply Linear Discriminant Analysis, to evaluate the amount of variance explained by each discriminant function and to plot the predictions with histograms. The code useful to compute the metrics for evaluating the LDA model is provided.

```
library(caret)
library(pROC)
library(MASS)
library(nortest)
library(ggplot2)

X_numeric <- dplyr::select_if(data, is.numeric)

# Anderson-Darling normality test
for (i in 1:dim(X_numeric)[2]){
  cat(colnames(X_numeric)[i], " --> ", ad.test(X_numeric[, i])$p.value, "\n"
      )
}

model_lda <- lda(Best_Position ~ . + Short_Passing:Ball_Control + Aggression
     :Interceptions + Short_Passing:Long_Passing + Finishing:Long_Shots +
     Acceleration:Sprint_Speed + Dribbling:Ball_Control + Free_Kick_Accuracy:
     Curve + Vision:Long_Passing + Agility:Dribbling + Agility:Ball_Control,
     data=training_set)
model_lda

# Canonic discrimination evaluation
model_lda$svd^2/sum(model_lda$svd^2) * 100

# Predicting the values for training dataset
train_bp_predicted <- predict(model_lda, newdata=training_set, type="
     response")
train_cf <- confusionMatrix(training_set$Best_Position, train_bp_predicted$
     class, mode="everything")

# Predicting the class for test dataset
test_bp_predicted <- predict(model_lda, newdata=test_set, type="class")
test_cf <- confusionMatrix(test_set$Best_Position, test_bp_predicted$class,
     mode="everything")

```

```r
roc <- multiclass.roc(test_set$Best_Position, as.numeric(test_bp_predicted$
    class))

# Metrics values for training phase
train_cf$overall[1] # Training Accuracy
# Metrics values for prediction phase
test_cf$overall[1] # Test Accuracy
roc$auc # AUC

# Plot predictions
for(i in 1:dim(test_bp_predicted$x)[2]){
  ggplot() +
  geom_histogram(aes(x=test_bp_predicted$x[, i],
                     y=stat(density)),
                     bins=20,
                     data=data.frame(test_bp_predicted)[1],
                     color="black",
                     fill=c("green")) +
  facet_grid(class ~ .) +
  labs(x=paste("LDA", i, sep=""))
}
```

Listing A.5: Linear Discriminant Analysis code

This code allows to apply Ridge Regression. Consider to set *set.seed(200)* to reproduce the results obtained in the thesis with Cross-Validation.

```r
library(caret)
library(pROC)
library(glmnet)

# New data structure for training set
X_training <- model.matrix(Best_Position ~ . + Short_Passing:Ball_Control +
    Aggression:Interceptions + Short_Passing:Long_Passing + Finishing:Long_
    Shots + Acceleration:Sprint_Speed + Dribbling:Ball_Control + Free_Kick_
    Accuracy:Curve + Vision:Long_Passing + Agility:Dribbling + Agility:Ball_
    Control, data=training_set)[,-1]
Y_training <- training_set$Best_Position
# New data structure for test set
X_test <- model.matrix(Best_Position ~ . + Short_Passing:Ball_Control +
    Aggression:Interceptions + Short_Passing:Long_Passing + Finishing:Long_
    Shots + Acceleration:Sprint_Speed + Dribbling:Ball_Control + Free_Kick_
    Accuracy:Curve + Vision:Long_Passing + Agility:Dribbling + Agility:Ball_
    Control, data=test_set)[,-1]
Y_test <- test_set$Best_Position

```

```r
model_ridge <- glmnet(X_training, Y_training, alpha=0, family="multinomial",
     type.measure="class", keep=TRUE, parallel=TRUE)
model_ridge

set.seed(200)
cv_model_ridge <- cv.glmnet(X_training, Y_training, alpha=0, family="
    multinomial", type.measure="class", keep=TRUE, parallel=TRUE)
min(cv_model_ridge$cvm) # Minimum mean cross-validated error
best_lambda_ridge <- cv_model_ridge$lambda.min

# Maximum explained deviance obtained from lambda.min
max(cv_model_ridge$glmnet.fit$dev.ratio)
# Coefficients for every variable for every level
coef(cv_model_ridge, cv_model_ridge$lambda.min)


train_bp_predicted <- predict(cv_model_ridge, newx=X_training, s="lambda.min
    ", type="class")
train_cf <- confusionMatrix(factor(train_bp_predicted, levels = c("CB", "RB"
    , "LB", "RWB", "LWB", "CDM", "CM", "RM", "LM", "CAM", "CF", "RW", "LW",
    "ST")), Y_training, mode="everything")

test_bp_predicted <- predict(cv_model_ridge, newx=X_test, s="lambda.min",
    type="class")
test_cf <- confusionMatrix(factor(test_bp_predicted, levels = c("CB", "RB",
    "LB", "RWB", "LWB", "CDM", "CM", "RM", "LM", "CAM", "CF", "RW", "LW", "
    ST")), Y_test, mode="everything")

class_probs_matrix_ridge <- predict(cv_model_ridge, newx=X_test, s="lambda.
    min", type="response")
dim(class_probs_matrix_ridge) <- c(dim(class_probs_matrix_ridge)[1], dim(
    class_probs_matrix_ridge)[2])
colnames(class_probs_matrix_ridge) <- c("CB", "RB", "LB", "RWB", "LWB", "CDM
    ", "CM", "RM", "LM", "CAM", "CF", "RW", "LW", "ST")
roc <- multiclass.roc(Y_test, class_probs_matrix_ridge)

tLL <- cv_model_ridge$glmnet.fit$nulldev - cv_model_ridge$glmnet.fit$nulldev
    *(1 - cv_model_ridge$glmnet.fit$dev.ratio)[which(cv_model_ridge$lambda
    == cv_model_ridge[["lambda.min"]])]
k <- cv_model_ridge$glmnet.fit$df[which(cv_model_ridge$lambda == cv_model_
    ridge[["lambda.min"]])]
n <- cv_model_ridge$glmnet.fit$nobs

# Metrics values for training phase
```

```
train_cf$overall[1] # Training Accuracy
AIC <- - tLL + 2*k + 2*k*(k + 1)/(n - k - 1) # AIC
AIC
BIC <- log(n)*k - tLL # BIC
BIC
# Metrics values for prediction phase
test_cf$overall[1] # Test Accuracy
roc$auc # AUC
```

Listing A.6: Ridge Regression code

This code allows to apply Lasso. Consider to set *set.seed(200)* to reproduce the results obtained in the thesis with Cross-Validation.

```
library(caret)
library(pROC)
library(glmnet)

# New data structure for training set
X_training <- model.matrix(Best_Position ~ . + Short_Passing:Ball_Control +
     Aggression:Interceptions + Short_Passing:Long_Passing + Finishing:Long_
     Shots + Acceleration:Sprint_Speed + Dribbling:Ball_Control + Free_Kick_
     Accuracy:Curve + Vision:Long_Passing + Agility:Dribbling + Agility:Ball_
     Control, data=training_set)[,-1]
Y_training <- training_set$Best_Position
# New data structure for test set
X_test <- model.matrix(Best_Position ~ . + Short_Passing:Ball_Control +
     Aggression:Interceptions + Short_Passing:Long_Passing + Finishing:Long_
     Shots + Acceleration:Sprint_Speed + Dribbling:Ball_Control + Free_Kick_
     Accuracy:Curve + Vision:Long_Passing + Agility:Dribbling + Agility:Ball_
     Control, data=test_set)[,-1]
Y_test <- test_set$Best_Position

model_lasso <- glmnet(X_training, Y_training, alpha=1, family="multinomial",
      type.measure="class", keep=TRUE, parallel=TRUE)
model_lasso

set.seed(200)
cv_model_lasso <- cv.glmnet(X_training, Y_training, alpha=1, family="
     multinomial", type.measure="class", keep=TRUE, parallel=TRUE)
cv_model_lasso$cvm[cv_model_lasso$index[2]] # Mean cross-validated error
     from lambda.1se
best_lambda_lasso <- cv_model_lasso$lambda.1se

# Maximum explained deviance obtained from lambda.1se
```

```
cv_model_lasso$glmnet.fit$dev.ratio[which(cv_model_lasso$lambda == cv_model_lasso$lambda.1se)]
# Coefficients for every variable for every level
coef(cv_model_lasso, cv_model_lasso$lambda.1se)
# Number of nonzero coefficients with lambda.1se
cv_model_lasso$nzero[which(cv_model_lasso$lambda == cv_model_lasso$lambda.1se)]


train_bp_predicted <- predict(cv_model_lasso, newx=X_training, s="lambda.1se", type="class")
train_cf <- confusionMatrix(factor(train_bp_predicted, levels = c("CB", "RB", "LB", "RWB", "LWB", "CDM", "CM", "RM", "LM", "CAM", "CF", "RW", "LW", "ST")), Y_training, mode="everything")

test_bp_predicted <- predict(cv_model_lasso, newx=X_test, s="lambda.1se", type="class")
test_cf <- confusionMatrix(factor(test_bp_predicted, levels = c("CB", "RB", "LB", "RWB", "LWB", "CDM", "CM", "RM", "LM", "CAM", "CF", "RW", "LW", "ST")), Y_test, mode="everything")

class_probs_matrix_ridge <- predict(cv_model_lasso, newx=X_test, s="lambda.1se", type="response")
dim(class_probs_matrix_ridge) <- c(dim(class_probs_matrix_ridge)[1], dim(class_probs_matrix_ridge)[2])
colnames(class_probs_matrix_ridge) <- c("CB", "RB", "LB", "RWB", "LWB", "CDM", "CM", "RM", "LM", "CAM", "CF", "RW", "LW", "ST")
roc <- multiclass.roc(Y_test, class_probs_matrix_ridge)

tLL <- cv_model_lasso$glmnet.fit$nulldev - cv_model_lasso$glmnet.fit$nulldev *(1 - cv_model_lasso$glmnet.fit$dev.ratio)[which(cv_model_lasso$lambda == cv_model_lasso[["lambda.1se"]])]
k <- cv_model_lasso$glmnet.fit$df[which(cv_model_lasso$lambda == cv_model_lasso[["lambda.1se"]])]
n <- cv_model_lasso$glmnet.fit$nobs

# Metrics values for training phase
train_cf$overall[1] # Training Accuracy
AIC <- - tLL + 2*k + 2*k*(k + 1)/(n - k - 1) # AIC
AIC
BIC <- log(n)*k - tLL # BIC
BIC
# Metrics values for prediction phase
test_cf$overall[1] # Test Accuracy
```

```
roc$auc # AUC
```

Listing A.7: Lasso code

B
Python code

This chapter includes the Python (Van Rossum and Drake, 2009) code produced during the writing of this thesis, in order to ensure the replicability of the obtained results.

B.1 The FIFA dataset

This code allows to apply web scraping technique on the first web page from https://sofifa.com/.

```python
from bs4 import BeautifulSoup
import requests
import time

def FirstPageWebScraping(url):
    if requests.get(f"{url}&offset=0").status_code == 200:
        page = requests.get(f"{url}&offset=0").text
        players_table = CatchWebPageTable(page)

        CatchFeaturesNames(players_table)
        CatchFeaturesValues(players_table)

        time.sleep(0.5)
    else:
        print(f"Error {requests.get('{url}&offset=0').status_code}")
```

Listing B.1: FirstPageWebScraping(url) method

This code allows to retrieve only the names of the attributes from the web page scraped.

```python
from bs4 import BeautifulSoup
import requests
import re

def CatchFeaturesNames(table):
    features_names = table.find_all("th")

    for row in features_names :
        cell = str(row)
        features_names_list.append(re.sub(re.compile('<.*?>'), '', cell))
```

Listing B.2: CatchFeaturesNames(players_table) method

This code allows to retrieve all the players' values for each attribute from the web page scraped.

```python
from bs4 import BeautifulSoup
import re
import pandas as pd

def CatchFeaturesValues(table):
    global results_features_values
    features_values = table.find_all("tr")

    for row in features_values :
        cells = re.sub(re.compile('Jun(.+?)</span>|<div class=\"tip\">(.+?)
            </span></div></div>'), '', str(row.find_all("td")))

        features_values_list.append(re.sub(re.compile('<.*?>'), '', cells))

    results_features_values = pd.DataFrame(features_values_list)
```

Listing B.3: CatchFeaturesValues(players_table) method

This code allows to apply web scraping technique on all the web pages following the first from https://sofifa.com/.

```python
from bs4 import BeautifulSoup
import requests
import time

def NextPagesWebScraping(url, offset):
    current_offset = offset

    while requests.get(f"{url}&offset={current_offset}").status_code == 200
        and current_offset <= 19980:
```

```
        page = requests.get(f"{url}&offset={current_offset}").text
        players_table = CatchWebPageTable(page)

        CatchFeaturesValues(players_table)

        current_offset = current_offset + 60
        time.sleep(0.5)

    if requests.get(f"{url}&offset={current_offset}").status_code != 200:
        print(f"Error {requests.get('{url}&offset={current_offset}').
            status_code}")
    else:
        print("All players have been downloaded.")
```

Listing B.4: NextPagesWebScraping(url, 60)) method

This code allows to manipulate and clean the dataset.

```
from bs4 import BeautifulSoup
import re

def ManipulateAndCleanData():
    # Split each row to cells
    cleaned_dataset = results_features_values[0].str.split(',', expand=True)

    # If the script donwloads the last column full of "None" values, drop it
    if cleaned_dataset.shape[1] > 53:
        delete_last_n_columns = cleaned_dataset.shape[1] - 53
        for i in range(delete_last_n_columns):
            cleaned_dataset.drop(cleaned_dataset.columns[-1], inplace=True,
                axis=1)

    # Set column names using table headers
    cleaned_dataset.columns = features_names_list[0 : len(
        features_names_list)]
    cleaned_dataset.rename(columns={'Height': 'Height_cm',
                                    'Weight': 'Weight_kg',
                                    'foot': 'Preferred_Foot',
                                    'BP': 'Best_Position',
                                    'Heading Accuracy': 'Heading_Accuracy',
                                    'Short Passing': 'Short_Passing',
                                    'FK Accuracy': 'Free_Kick_Accuracy',
                                    'Long Passing': 'Long_Passing',
                                    'Ball Control': 'Ball_Control',
                                    'Sprint Speed': 'Sprint_Speed',
                                    'Shot Power': 'Shot_Power',
                                        'Long Shots': 'Long_Shots',
                                        'Marking': 'Defensive_Awareness',
                                        'Standing Tackle': 'Standing_Tackle',
                                        'Sliding Tackle': 'Sliding_Tackle',
                                        'GK Diving': 'GK_Diving',
                                        'GK Handling': 'GK_Handling',
                                        'GK Kicking': 'GK_Kicking',
                                        'GK Positioning': 'GK_Positioning',
                                        'GK Reflexes': 'GK_Reflexes',
                                        'W/F': 'Weak_Foot',
                                        'SM': 'Skill_Moves',
                                        'A/W': 'Attacking_Work_Rate',
                                        'D/W': 'Defensive_Work_Rate',
                                        'PAC': 'Pace_Diving',
                                        'SHO': 'Shooting_Handling',
                                        'PAS': 'Passing_Kicking',
                                        'DRI': 'Dribbling_Reflexes',
                                        'DEF': 'Defending_Pace',
                                        'PHY': 'Physical_Position'},
                                inplace=True)

    # Drop the first, "Team & Contract" and "Skill_Moves" columns because
        unmeaningful
    cleaned_dataset.drop(cleaned_dataset.columns[0], inplace=True, axis=1)
    cleaned_dataset.drop(columns = ['Team &amp; Contract'], inplace=True,
        axis=1)
    cleaned_dataset.drop(columns = ['Skill_Moves'], inplace=True, axis=1)

    # Fix typos or character errors
    cleaned_dataset["Name"] = cleaned_dataset["Name"].str.replace(r"(\n|LW|
        ST|RW|LF|CF|RF|CAM|LM|CM|RM|CDM|LWB|LB|CB|RB|RWB|GK)", "", regex=
        True)
    cleaned_dataset["Name"] = cleaned_dataset["Name"].str.replace(r"[0-9\n]"
        , "", regex=True)
    cleaned_dataset["Height_cm"] = cleaned_dataset["Height_cm"].str.replace(
        r"(\n|cm)", "", regex=True)
    cleaned_dataset["Weight_kg"] = cleaned_dataset["Weight_kg"].str.replace(
        "kg", "", regex=False)
    cleaned_dataset["Physical_Position"] = cleaned_dataset["
        Physical_Position"].str.replace("]", "", regex=False)
    for i in range(0, cleaned_dataset.shape[1]):
        cleaned_dataset.iloc[:, i] = cleaned_dataset.iloc[:, i].str.replace(
            "N/A", "", regex=False)

```

```python
    # Remove spaces
    for i in range(0, len(cleaned_dataset.columns)):
        cleaned_dataset.iloc[:, i] = cleaned_dataset.iloc[:, i].str.strip()

    # Remove every row which contains only Nan values or empty cells
    index_of_NaN_rows = []
    for i in range(0, cleaned_dataset.shape[0]):
        if cleaned_dataset.iloc[i, :].isnull().values.any() |
            cleaned_dataset.iloc[i, :].eq("").sum() > 0:
            index_of_NaN_rows.append(i)
    for i in range(0, len(index_of_NaN_rows)):
        cleaned_dataset.drop(labels=index_of_NaN_rows[i], inplace=True, axis
            =0)

    return cleaned_dataset
```

Listing B.5: ManipulateAndCleanData() method

This code allows to generate the dataset in CSV format in the desired *path*.

```python
def GenerateCSVDataset(dataset):
    dataset.to_csv("path/FIFA22_Dataset.csv", index=False, encoding="UTF-8",
        na_rep='NA', mode="a")
```

Listing B.6: GenerateCSVDataset(cleaned_dataset) method

B.2 Machine Learning Results

This code allows to adapt the dataset for the Machine Learning techniques. Upload the *.RData* file in which both the training set and test set have been saved in a specific *path*.

```python
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import pandas as pd

def setCategoricalVariables(training_set, test_set):
  # Set categorical variables - TRAINING SET
  training_set["Best_Position"] = training_set["Best_Position"].cat.
      set_categories(['CB', 'RB', 'LB', 'RWB', 'LWB', 'CDM', 'CM', 'RM', 'LM
      ', 'CAM', 'CF', 'RW', 'LW', 'ST'], ordered=True)
  training_set["Weak_Foot"] = training_set["Weak_Foot"].cat.set_categories([
      '1', '2', '3', '4', '5'], ordered=True)
  training_set["Attacking_Work_Rate"] = training_set["Attacking_Work_Rate"].
      cat.set_categories(['Low', 'Medium', 'High'], ordered=True)
  training_set["Defensive_Work_Rate"] = training_set["Defensive_Work_Rate"].
      cat.set_categories(['Low', 'Medium', 'High'], ordered=True)

  # Set categorical variables - TEST SET
  test_set["Best_Position"] = test_set["Best_Position"].cat.set_categories([
      'CB', 'RB', 'LB', 'RWB', 'LWB', 'CDM', 'CM', 'RM', 'LM', 'CAM', 'CF',
      'RW', 'LW', 'ST'], ordered=True)
  test_set["Weak_Foot"] = test_set["Weak_Foot"].cat.set_categories(['1', '2'
      , '3', '4', '5'], ordered=True)
  test_set["Attacking_Work_Rate"] = test_set["Attacking_Work_Rate"].cat.
      set_categories(['Low', 'Medium', 'High'], ordered=True)
  test_set["Defensive_Work_Rate"] = test_set["Defensive_Work_Rate"].cat.
      set_categories(['Low', 'Medium', 'High'], ordered=True)

  return training_set, test_set

def applyOneHotEncoding(training_set, test_set):
  encoder = OneHotEncoder(dtype='int32', handle_unknown='ignore')

  # One Hot Encoding - TRAINING SET
  predictor_encoded = pd.DataFrame(encoder.fit_transform(training_set[['
      Preferred_Foot']]).toarray())
  predictor_encoded.columns = ['Preferred_Foot_Left', 'Preferred_Foot_Right'
      ]
  training_set = training_set.join(predictor_encoded)
  training_set.drop('Preferred_Foot', axis=1, inplace=True)
  predictor_encoded = pd.DataFrame(encoder.fit_transform(training_set[['
      Weak_Foot']]).toarray())
  predictor_encoded.columns = ['Weak_Foot_1', 'Weak_Foot_2', 'Weak_Foot_3',
      'Weak_Foot_4', 'Weak_Foot_5']
  training_set = training_set.join(predictor_encoded)
  training_set.drop('Weak_Foot', axis=1, inplace=True)
  predictor_encoded = pd.DataFrame(encoder.fit_transform(training_set[['
      Body_Type']]).toarray())
  predictor_encoded.columns = ['Body_Type_Lean (170-)', 'Body_Type_Lean
      (170-185)', 'Body_Type_Lean (185+)',
                              'Body_Type_Normal (170-)', 'Body_Type_Normal
                                  (170-185)', 'Body_Type_Normal (185+)',
                              'Body_Type_Stocky (170-)', 'Body_Type_Stocky
                                  (170-185)', 'Body_Type_Stocky (185+)',
                              'Body_Type_Unique']
  training_set = training_set.join(predictor_encoded)
  training_set.drop('Body_Type', axis=1, inplace=True)
  predictor_encoded = pd.DataFrame(encoder.fit_transform(training_set[['
      Attacking_Work_Rate']]).toarray())
```

```
predictor_encoded.columns = ['Attacking_Work_Rate_High', 'Attacking_Work_Rate_Low', 'Attacking_Work_Rate_Medium']
training_set = training_set.join(predictor_encoded)
training_set.drop('Attacking_Work_Rate', axis=1, inplace=True)
predictor_encoded = pd.DataFrame(encoder.fit_transform(training_set[['Defensive_Work_Rate']]).toarray())
predictor_encoded.columns = ['Defensive_Work_Rate_High', 'Defensive_Work_Rate_Low', 'Defensive_Work_Rate_Medium']
training_set = training_set.join(predictor_encoded)
training_set.drop('Defensive_Work_Rate', axis=1, inplace=True)
del predictor_encoded

# One Hot Encoding - TEST SET
predictor_encoded = pd.DataFrame(encoder.fit_transform(test_set[['Preferred_Foot']]).toarray())
predictor_encoded.columns = ['Preferred_Foot_Left', 'Preferred_Foot_Right']
test_set = test_set.join(predictor_encoded)
test_set.drop('Preferred_Foot', axis=1, inplace=True)
predictor_encoded = pd.DataFrame(encoder.fit_transform(test_set[['Weak_Foot']]).toarray())
predictor_encoded.columns = ['Weak_Foot_1', 'Weak_Foot_2', 'Weak_Foot_3', 'Weak_Foot_4', 'Weak_Foot_5']
test_set = test_set.join(predictor_encoded)
test_set.drop('Weak_Foot', axis=1, inplace=True)
predictor_encoded = pd.DataFrame(encoder.fit_transform(test_set[['Body_Type']]).toarray())
predictor_encoded.columns = ['Body_Type_Lean (170-)', 'Body_Type_Lean (170-185)', 'Body_Type_Lean (185+)',
                             'Body_Type_Normal (170-)', 'Body_Type_Normal (170-185)', 'Body_Type_Normal (185+)',
                             'Body_Type_Stocky (170-)', 'Body_Type_Stocky (170-185)', 'Body_Type_Stocky (185+)',
                             'Body_Type_Unique']
test_set = test_set.join(predictor_encoded)
test_set.drop('Body_Type', axis=1, inplace=True)
predictor_encoded = pd.DataFrame(encoder.fit_transform(test_set[['Attacking_Work_Rate']]).toarray())
predictor_encoded.columns = ['Attacking_Work_Rate_High', 'Attacking_Work_Rate_Low', 'Attacking_Work_Rate_Medium']
test_set = test_set.join(predictor_encoded)
test_set.drop('Attacking_Work_Rate', axis=1, inplace=True)
predictor_encoded = pd.DataFrame(encoder.fit_transform(test_set[['Defensive_Work_Rate']]).toarray())
```

```
  predictor_encoded.columns = ['Defensive_Work_Rate_High', '
      Defensive_Work_Rate_Low', 'Defensive_Work_Rate_Medium']
  test_set = test_set.join(predictor_encoded)
  test_set.drop('Defensive_Work_Rate', axis=1, inplace=True)
  del predictor_encoded

  return training_set, test_set

def insertInteraction(training_set, test_set):
  training_set['Short_Passing:Ball_Control'] = training_set['Short_Passing']
       * training_set['Ball_Control']
  test_set['Short_Passing:Ball_Control'] = test_set['Short_Passing'] *
      test_set['Ball_Control']
  training_set['Aggression:Interceptions'] = training_set['Aggression'] *
      training_set['Interceptions']
  test_set['Aggression:Interceptions'] = test_set['Aggression'] * test_set['
      Interceptions']
  training_set['Short_Passing:Long_Passing'] = training_set['Short_Passing']
       * training_set['Long_Passing']
  test_set['Short_Passing:Long_Passing'] = test_set['Short_Passing'] *
      test_set['Long_Passing']
  training_set['Finishing:Long_Shots'] = training_set['Finishing'] *
      training_set['Long_Shots']
  test_set['Finishing:Long_Shots'] = test_set['Finishing'] * test_set['
      Long_Shots']
  training_set['Acceleration:Sprint_Speed'] = training_set['Acceleration'] *
       training_set['Sprint_Speed']
  test_set['Acceleration:Sprint_Speed'] = test_set['Acceleration'] *
      test_set['Sprint_Speed']
  training_set['Dribbling:Ball_Control'] = training_set['Dribbling'] *
      training_set['Ball_Control']
  test_set['Dribbling:Ball_Control'] = test_set['Dribbling'] * test_set['
      Ball_Control']
  training_set['Free_Kick_Accuracy:Curve'] = training_set['
      Free_Kick_Accuracy'] * training_set['Curve']
  test_set['Free_Kick_Accuracy:Curve'] = test_set['Free_Kick_Accuracy'] *
      test_set['Curve']
  training_set['Vision:Long_Passing'] = training_set['Vision'] *
      training_set['Long_Passing']
  test_set['Vision:Long_Passing'] = test_set['Vision'] * test_set['
      Long_Passing']
  training_set['Agility:Dribbling'] = training_set['Agility'] * training_set
      ['Dribbling']
```

```python
    test_set['Agility:Dribbling'] = test_set['Agility'] * test_set['Dribbling']
    training_set['Agility:Ball_Control'] = training_set['Agility'] * training_set['Ball_Control']
    test_set['Agility:Ball_Control'] = test_set['Agility'] * test_set['Ball_Control']

    return training_set, test_set

# Recover RData file
r_file = pyreadr.read_r('path/dataset_split.RData')
training_set = r_file["training_set"]
test_set = r_file["test_set"]

training_set, test_set = setCategoricalVariables(training_set, test_set)
training_set, test_set = applyOneHotEncoding(training_set, test_set)
training_set, test_set = insertInteraction(training_set, test_set)

# Split dataset
X_training = training_set.drop('Best_Position', axis=1)
Y_training = training_set['Best_Position']
X_test = test_set.drop('Best_Position', axis=1)
Y_test = test_set['Best_Position']

scaler = StandardScaler().fit(X_training)
X_training_standardized = pd.DataFrame(scaler.transform(X_training))
X_test_standardized = pd.DataFrame(scaler.transform(X_test))
```

Listing B.7: Dataset adaptation for Machine Learning techniques

This code allows to apply k-Fold Cross-Validation with the *GridSearchCV* function. Consider to set *random.state=42* to reproduce the results obtained in the thesis with Cross-Validation.

```python
from sklearn.model_selection import GridSearchCV, StratifiedKFold

def applyGridSearchCV(estimator, param_grid, X_training, Y_training, X_test, Y_test):
    gscv = GridSearchCV(estimator, param_grid, cv=StratifiedKFold(n_splits=10, shuffle=True, random_state=42), verbose=5, return_train_score=True)
    gscv.fit(X_training, Y_training)

    print('Best model after hyper-parameter tuning: ', gscv.best_estimator_)
    print('Parameters for the best model: ', gscv.best_params_)
    print('Mean score for the best model: ', gscv.best_score_)

```

```
  print("GridSearchCV Training accuracy:", gscv.score(X_training, Y_training
      ))
  print("GridSearchCV Test accuracy:", gscv.score(X_test, Y_test))

  return gscv
```

Listing B.8: k-Fold Cross-Validation with GridSearchCV

This code allows to print the metrics used for Machine Learning techniques, namely, Training Accuracy, Test Accuracy and AUC. The confusion matrix for both the training phase and the prediction phase is computed. The sensitivity values for each player's position are shown.

```
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, roc_auc_score

def printMetrics(fitted_model, X_training, Y_training, X_test, Y_test):
  train_bp_predicted = fitted_model.predict(X_training)
  test_bp_predicted = fitted_model.predict(X_test)
  test_bp_predicted_probs = fitted_model.predict_proba(X_test) # Probability
      values instead of labels

  train_cf = confusion_matrix(Y_training, train_bp_predicted)
  train_report = classification_report(Y_training, train_bp_predicted)
  training_accuracy = accuracy_score(Y_training, train_bp_predicted)
  print("TRAINING PHASE - RESULTS")
  print("Confusion matrix:")
  print(train_cf)
  print("Classification metrics:")
  print(train_report)
  print("Accuracy: ", training_accuracy, "\n\n")

  test_cf = confusion_matrix(Y_test, test_bp_predicted)
  test_report = classification_report(Y_test, test_bp_predicted)
  test_accuracy = accuracy_score(Y_test, test_bp_predicted)
  print("TEST PHASE - RESULTS")
  print("Confusion matrix:")
  print(test_cf)
  print("Classification metrics:")
  print(test_report)
  print("Accuracy: ", test_accuracy, "\n\n")
  auc = roc_auc_score(Y_test, test_bp_predicted_probs, multi_class='ovr')
  print('AUC: ', auc)
```

Listing B.9: Machine Learning metrics

This code allows to apply the Decision Tree technique. Consider to set *random.state=42* to reproduce the results obtained in the thesis.

```python
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.tree import DecisionTreeClassifier

def applyDecisionTree(X_training, Y_training, X_test, Y_test):
  dtc = DecisionTreeClassifier(random_state=42)
  dtc = dtc.fit(X_training, Y_training)
  y_pred = dtc.predict(X_test)
  print("Training accuracy:", dtc.score(X_training, Y_training))
  print("Test Accuracy:", accuracy_score(Y_test, y_pred))

  estimator = DecisionTreeClassifier(random_state=42)
  param_grid = {'criterion': ['gini', 'entropy'],
                'max_depth': np.arange(1,31)}
  dtc_gscv = applyGridSearchCV(estimator, param_grid, X_training, Y_training
      , X_test, Y_test)

  printMetrics(dtc_gscv, X_training, Y_training, X_test, Y_test)

applyDecisionTree(X_training, Y_training, X_test, Y_test)
```

Listing B.10: Decision Tree code

This code allows to apply the Decision Tree technique with Minimal Cost-Complexity Pruning algorithm. Consider to set *random.state=42* to reproduce the results obtained in the thesis.

```python
import numpy as np
from sklearn.tree import DecisionTreeClassifier

def applyPruningDecisionTree(X_training, Y_training, X_test, Y_test):
  dtc = DecisionTreeClassifier(criterion='entropy', max_depth=11,
      random_state=42)
  path = dtc.cost_complexity_pruning_path(X_training, Y_training)
  ccp_alphas, impurities = path.ccp_alphas, path.impurities

  dtcs_pruned = []
  for ccp_alpha in ccp_alphas:
      dtc_pruned = DecisionTreeClassifier(criterion='entropy', max_depth=11,
          random_state=42, ccp_alpha=ccp_alpha)
      dtc_pruned.fit(X_training, Y_training)
      dtcs_pruned.append(dtc_pruned)

```

```python
    train_scores = [dtc.score(X_training, Y_training) for dtc in dtcs_pruned]
    test_scores = [dtc.score(X_test, Y_test) for dtc in dtcs_pruned]

    best_ccp_alpha = ccp_alphas[np.argmax(test_scores)]
    dtc_pruned_best = DecisionTreeClassifier(criterion='entropy', max_depth
        =11, random_state=42, ccp_alpha=best_ccp_alpha)
    dtc_pruned_best.fit(X_training, Y_training)
    printMetrics(dtc_pruned_best, X_training, Y_training, X_test, Y_test)

applyPruningDecisionTree(X_training, Y_training, X_test, Y_test)
```

Listing B.11: Pruning Decision Tree code

This code allows to apply the Random Forest technique. Consider to set *random.state=42* to reproduce the results obtained in the thesis.

```python
import numpy as np
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier

def applyRandomForest(X_training, Y_training, X_test, Y_test):
    rfc = RandomForestClassifier(n_estimators=100, random_state=42)
    rfc.fit(X_training, Y_training)
    y_pred = rfc.predict(X_test)
    print("Training accuracy:", rfc.score(X_training, Y_training))
    print("Test Accuracy:", accuracy_score(Y_test, y_pred))

    estimator = RandomForestClassifier(random_state=42)
    param_grid = {'n_estimators': np.arange(1, 101),
                  'criterion': ['gini', 'entropy'],
                  'max_depth': np.arange(1,31),
                  'max_features': ['sqrt', 'log2']}
    rfc_gscv = applyGridSearchCV(estimator, param_grid, X_training, Y_training
        , X_test, Y_test)

    printMetrics(rfc_gscv, X_training, Y_training, X_test, Y_test)

applyRandomForest(X_training, Y_training, X_test, Y_test)
```

Listing B.12: Random Forest code

This code allows to apply the KNN technique.

```python
import numpy as np
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier

```

```
def applyKNearestNeighbour(X_training_scaled, Y_training, X_test_scaled,
    Y_test):
  knn = KNeighborsClassifier()
  knn.fit(X_training_scaled, Y_training)
  y_pred = knn.predict(X_test_scaled)
  print("Training accuracy:", knn.score(X_training_scaled, Y_training))
  print("Test Accuracy:", accuracy_score(Y_test, y_pred))

  estimator = KNeighborsClassifier()
  param_grid = {'n_neighbors': np.arange(1,101),
                'weights': ['uniform', 'distance']}
  knn_gscv = applyGridSearchCV(estimator, param_grid, X_training_scaled,
      Y_training, X_test_scaled, Y_test)

  printMetrics(knn_gscv, X_training_scaled, Y_training, X_test_scaled,
      Y_test)

applyKNearestNeighbour(X_training_standardized, Y_training,
    X_test_standardized, Y_test)
```

Listing B.13: K-Nearest Neighbour code

This code allows to apply the Naive Bayes technique with *MultinomialNB* function and *GaussianNB* function.

```
from sklearn.naive_bayes import GaussianNB, CategoricalNB

def applyMultinomialNaiveBayes(X_training, Y_training, X_test, Y_test):
  nb = MultinomialNB()
  nb.fit(X_training, Y_training)
  y_pred = nb.predict(X_test)
  print("Training accuracy:", nb.score(X_training, Y_training))
  print("Test Accuracy:", accuracy_score(Y_test, y_pred))

  printMetrics(nb, X_training, Y_training, X_test, Y_test)

def applyGaussianNaiveBayes(X_training, Y_training, X_test, Y_test):
  nb = GaussianNB()
  nb.fit(X_training, Y_training)
  y_pred = nb.predict(X_test)
  print("Training accuracy:", nb.score(X_training, Y_training))
  print("Test Accuracy:", accuracy_score(Y_test, y_pred))

  printMetrics(nb, X_training, Y_training, X_test, Y_test)

applyMultinomialNaiveBayes(X_training, Y_training, X_test, Y_test)
```

```
applyGaussianNaiveBayes(X_training, Y_training, X_test, Y_test)
```

Listing B.14: Naive Bayes code

This code allows to apply the SVM technique. Consider to set *random.state=42* to reproduce the results obtained in the thesis.

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.svm import SVC

def applySupportVectorMachine(X_training_scaled, Y_training, X_test_scaled,
    Y_test):
  print("SVM with linear kernel -> ")
  svc_linear = SVC(kernel='linear', random_state=42, probability=True)
  svc_linear.fit(X_training_scaled, Y_training)
  y_pred_linear = svc_linear.predict(X_test_scaled)
  print("Training accuracy:", svc_linear.score(X_training_scaled, Y_training
    ))
  print("Test Accuracy:", accuracy_score(Y_test, y_pred_linear))

  estimator_linear = SVC(kernel='linear', random_state=42, probability=True)
  param_grid_linear = {'C': [0.1, 0.5, 1]}
  svc_gscv_linear = applyGridSearchCV(estimator_linear, param_grid_linear,
    X_training_scaled, Y_training, X_test_scaled, Y_test)

  printMetrics(svc_gscv_linear, X_training_scaled, Y_training, X_test_scaled
    , Y_test)


  print("\nSVM with poly kernel -> ")
  svc_poly = SVC(kernel='poly', gamma=1, coef0=1, random_state=42,
    probability=True)
  svc_poly.fit(X_training_scaled, Y_training)
  y_pred_poly = svc_poly.predict(X_test_scaled)
  print("Training accuracy:", svc_poly.score(X_training_scaled, Y_training))
  print("Test Accuracy:", accuracy_score(Y_test, y_pred_poly))

  estimator_poly = SVC(kernel='poly', gamma=1, coef0=1, random_state=42,
    probability=True)
  param_grid_poly = {'C': [0.1, 0.5, 1],
                     'degree': [1, 2, 3]}
  svc_gscv_poly = applyGridSearchCV(estimator_poly, param_grid_poly,
    X_training_scaled, Y_training, X_test_scaled, Y_test)

  printMetrics(svc_gscv_poly, X_training_scaled, Y_training, X_test_scaled,
    Y_test)


    print("\nSVM with rbf kernel -> ")
    svc_rbf = SVC(kernel='rbf', random_state=42, probability=True)
    svc_rbf.fit(X_training_scaled, Y_training)
    y_pred_rbf = svc_rbf.predict(X_test_scaled)
    print("Training accuracy:", svc_rbf.score(X_training_scaled, Y_training))
    print("Test Accuracy:", accuracy_score(Y_test, y_pred_rbf))

    estimator_rbf = SVC(kernel='rbf', random_state=42, probability=True)
    param_grid_rbf = {'C': [0.1, 0.5, 1],
                      'gamma': [0.1, 0.01, 0.001]}
    svc_gscv_rbf = applyGridSearchCV(estimator_rbf, param_grid_rbf,
        X_training_scaled, Y_training, X_test_scaled, Y_test)

    printMetrics(svc_gscv_rbf, X_training_scaled, Y_training, X_test_scaled,
        Y_test)

applySupportVectorMachine(X_training_standardized, Y_training,
    X_test_standardized, Y_test)
```

Listing B.15: Support Vector Machine code

Glossary

AIC: acronym of *Akaike Information Criterion*, it is an estimator of prediction error used in data mining techniques available in Section § 4.3. 51, 159

AUC: acronym of *Area Under the ROC Curve*, it is a metric for computing the prediction goodness used both in data mining techniques and machine learning techniques available in Section § 5.1.2. 58, 159

BIC: acronym of *Bayesian Information Criterion*, it is an estimator of prediction error used in data mining techniques available in Section § 4.3. 51, 159

CSV: acronym of *Comma Separated Values*, it is a text-based data format that separates fields with a comma and ends with a line break , resulting in a table structured format. 22, 159

EA Sports: acronym of *Electronic Arts Sports*, it is a division of Electronic Arts that develops and publishes sports video games. Formerly a marketing gimmick of Electronic Arts, in which they tried to imitate real-life sports networks by calling themselves the "EA Sports Network" (EASN), it soon grew up to become a sub-label on its own, releasing game series such as *FIFA*, *NHL*, *NBA Live*, *F1* and *Madden NFL*. 8, 159

FIFA: acronym of *Fédération Internationale de Football Association*, it is an international governing body of association football, beach football and futsal. Headquartered in Zürich, Switzerland, its membership now comprises 211 national associations. These national associations must each also be members of one of the six regional confederations into which the world is divided: Africa, Asia, Europe, North & Central America and the Caribbean, Oceania and South America. FIFA name appears also in the most famous football video game, as *FIFA 22* cited plenty of times in this thesis, and the game is developed by EA Sports. 2, 159

FUT: acronym of *FIFA Ultimate Team*, it is a game mode that lets you build your dream squad in *FIFA 22* video game. 14, 159

HTML: acronym of *HyperText Markup Language*, it is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as CSS (Cascading Style Sheets) and scripting languages such as JavaScript. 22, 159

ID3: acronym of *Iterative Dichotomiser 3*, it is the algorithm invented by Ross Quinlan for choosing the best attribute of a node while building a decision tree, a machine learning algorithm available in Section § 6.2. ID3 uses a top-down approach and the acronym suggests that the algorithm repeatedly splits attributes into two or more groups at each step, until reaching the end of the decision tree. 82, 159

IDE: acronym of *Integrated Development Environment*, it is a software application for developing code through a graphical user interface. The standard tools provided are a source code editor, build automation tools, a debugger, a compiler and an integrated version control system. 5, 159

KNN: acronym of *K-Nearest Neighbour*, it is a machine learning technique available in Section § 6.4. 83, 159

Lasso: acronym of *Least Absolute Shrinkage and Selection Operator*, it is a regularization data mining technique available in Section § 4.5.2. 53, 159

LDA: acronym of *Linear Discriminant Analysis*, it is a data mining technique available in Section § 4.4. 51, 159

OS: acronym of *Operating System*, it is a system software that, after being initially loaded into the computer by a boot program, manages computer hardware, software resources, and provides common services for computer programs. 4, 159

Python: it is a high-level, general-purpose programming language, which is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a ”batteries included” language due to its comprehensive standard library. 7

ROC: acronym of *Receiver Operating Characteristics*, it is a graphical tool useful to understand the trade-off between sensitivity and 1-specificity. AUC metric is strictly correlated to ROC. 58, 159

SVM: acronym of *Support Vector Machine*, it is a machine learning technique available in Section § 6.6. 86, 159

ToS: acronym of *Terms of Service*, is a type of document stating details about what a service provider is responsible for as well as user obligations that must be adhered to for continuation of the service. Users that don't follow the rules specified in a ToS are subject to termination. Many websites and applications publish their terms of service. Terms of service should include user rights and responsibilities. The ToS should also be transparent about all activities of the service that have significance for users, such as details of what the service does with user data as well as how the service maintains user privacy and security. 26, 159

URL: acronym of *Uniform Resource Locator*, is a sequence of characters that uniquely identifies the address of a web resource. 22, 160

UTF-8: acronym of *Universal Transformation Format-8*, is a variable-width character encoding used for electronic communication. Defined by the Unicode Standard, it is capable of encoding all 1,112,064 valid character code points in Unicode using one to four one-byte (8-bit) code units. 25, 160

Web Scraping: it is a technique used to collect information from web sites. Generally, this is done with software that simulates human Web surfing to collect specified bits of information from different web sites. It is used as a component of applications used for web indexing, web mining and data mining. 22

WEKA: acronym of *Waikato Environment for Knowledge Analysis*, it is a free software for machine learning which contains data analysis and predictive modeling tools. WEKA is developed and maintained by the University of Waikato, in New Zealand. 3, 160

XML: acronym of *eXtensible Markup Language*, it is a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It defines a set of rules for

encoding documents in a format that is both human-readable and machine-readable.
22, 160

Acronyms

AIC: Akaike Information Criterion. 155

AUC: Area Under the ROC Curve. 155

BIC: Bayesian Information Criterion. 155

CSV: Comma Separated Values. 155

EA Sports: Electronic Arts Sports. 155

FIFA: Fédération Internationale de Football Association. 155

FUT: FIFA Ultimate Team. 155

HTML: HyperText Markup Language. 156

ID3: Iterative Dichotomiser 3. 156

IDE: Integrated Development Environment. 156

KNN: K-Nearest Neighbour. 156

Lasso: Least Absolute Shrinkage and Selection Operator. 156

LDA: Linear Discriminant Analysis. 156

OS: Operating System. 156

ROC: Receiver Operating Characteristics. 157

SVM: Support Vector Machine. 157

ToS: Terms of Service. 157

URL: Uniform Resource Locator. 157

UTF-8: Universal Transformation Format-8. 157

WEKA: Waikato Environment for Knowledge Analysis. 157

XML: eXtensible Markup Language. 157

Bibliography

Agresti, Alan (2013). *Categorical data analysis*. Third Edition. Wiley.

Azzalini, Adelchi and Bruno Scarpa (2012). *Data Analysis and Data Mining: An introduction*. Second Edition. Oxford University Press.

Bazmara, Mohammad (Sept. 2014). A Novel Fuzzy Approach for Determining Best Position of Soccer Players. Vol. **09**. *International Journal of Intelligent Systems and Applications*, pp. 62–67. URL: https://www.researchgate.net/publication/264549745_A_Novel_Fuzzy_Approach_for_Determining_Best_Position_of_Soccer_Players.

Bazmara, Mohammad and Shahram Jafari (Apr. 2013). K Nearest Neighbor Algorithm for Finding Soccer Talent. Vol. **3**. *Journal of Basic and Applied Scientific Research*, pp. 981–986. URL: https://www.researchgate.net/publication/237080861_K_Nearest_Neighbor_Algorithm_for_Finding_Soccer_Talent.

Bosu Babu, S et al. (May 2022). Predicting football player's position. Vol. **4**. *International Research Journal of Modernization in Engineering Technology and Science*. URL: https://www.irjmets.com/uploadedfiles/paper/issue_5_may_2022/23815/final/fin_irjmets1653665116.pdf.

Hand, David J. and Robert J. Till (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. Vol. **45**. *Machine Learning*, pp. 171–186. URL: https://link.springer.com/article/10.1023/A:1010920819831.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The elements of statistical learning: data mining, inference and prediction*. Second Edition. Springer.

James, Gareth et al. (2021). *An Introduction to Statistical Learning: with Applications in R*. Third Edition. Springer.

Kabacoff, Rob (2018). *Data Visualization with R*. Quantitative Analysis Center (QAC), Wesleyan University.

Lantz, Brett (2015). *Machine Learning with R*. Second Edition. Packt Publishing.

Lewis, Chris and Noah Wardrip-Fruin (June 2010). Mining Game Statistics from Web Services: A World of Warcraft Armory case study. *University of California, Santa Cruz*. URL: https://dl.acm.org/doi/pdf/10.1145/1822348.1822362?casa_token=UaOhftMGXnMAAAAA:p417v5pKY6_i40nWHCFCfJfCjKuT7ffnagGOVX9Nwomou_1jtuJ1kIh1NC4GlC2AQmVpNFAQcqxs.

Mitchell, Tom M (1997). *Machine Learning*. McGraw-Hill New York.

Pace, Luigi and Alessandra Salvan (1997). Principles of Statistical Inference from a Neo-Fisherian Perspective. Vol. **4**. World Scientific Publishing Company. *Advanced Series On Statistical Science And Applied Probability*.

R Core Team (2022). R: A Language and Environment for Statistical Computing. *R Foundation for Statistical Computing*. Vienna, Austria. URL: https://www.R-project.org/.

Razali, Nazim et al. (2017). Predicting Player Position for Talent Identification in Association Football. Vol. **226**. *IOP Conf. Series: Materials Science and Engineering*. URL: https://iopscience.iop.org/article/10.1088/1757-899X/226/1/012087/pdf.

Van Rossum, Guido and Fred L. Drake (2009). *Python 3 Reference Manual*. CreateSpace.

Wickham, Hadley (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

Zixue, Zeng and Bingyu Pan (2021). A Machine Learning Model to Predict Player's Positions based on Performance. Vol. **1**. *In Proceedings of the 9th International Conference on Sport Sciences Research and Technology Support*, pp. 36–42. URL: https://www.scitepress.org/Papers/2021/106533/106533.pdf.

Sitography

Central Attacking Midfielder. URL: https://www.fifplay.com/encyclopedia/central-attacking-midfielder/.

Central Defensive Midfielder. URL: https://www.fifplay.com/encyclopedia/central-defensive-midfielder/.

Central Midfielder. URL: https://www.fifplay.com/encyclopedia/central-midfielder/.

Centre Back. URL: https://www.fifplay.com/encyclopedia/centre-back/.

Centre Forward. URL: https://www.fifplay.com/encyclopedia/centre-forward/.

Data structure DataFrame. Python data structure for creating two-dimensional tabular data. URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

Decision Trees mathematical formulation. URL: https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation.

FIFA 22 Player Attributes. URL: https://www.fifplay.com/fifa-22-player-attributes/.

FIFA 22 Ultimate Team. Guide to play FIFA 22 Ultimate Team (FUT). By Electronic Arts Inc. URL: https://help.ea.com/nz/help/fifa/fifa-ultimate-team-fut/#:~:text=FIFA%5C%20Ultimate%5C%20Team%5C%20(FUT)%5C%20is,your%5C%20mark%5C%20on%5C%20the%5C%20pitch..

FIFA 22 Work Rates. Work Rates for FIFA 22 Ultimate Team. By FIFAUTeam. URL: https://fifauteam.com/work-rates-fifa-22/.

FIFA player ratings. FIFA player ratings explained: How are the card number stats decided? By Ronan Murphy, Goal. URL: https://www.goal.com/en-sa/news/fifa-player-ratings-explained-how-are-the-card-number-stats/1hszd2fgr7wgf1n2b2yjdpgynu.

FIFPlay Player Attributes. URL: https://www.fifplay.com/encyclopedia/player-attributes/.

Football Analytics. The Growing Importance of Football Analytics. URL: https://soccerment.com/the-importance-of-football-analytics/.

Football Talent Scout. URL: https://fifa-talentscout.ea.com/TalentScout/WelcomeTS.aspx.

Formation. URL: https://www.fifplay.com/encyclopedia/formation/.

Function ad.test. R fucntion for computing Anderson-Darling normality test. URL: https://www.rdocumentation.org/packages/nortest/versions/1.0-4/topics/ad.test.

Function anova. R function for computing analysis of variance for fitted models. URL: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/anova.

Function confusion_matrix. Python function for creating a confusion matrix. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

Function confusionMatrix. R function for creating a confusion matrix. URL: https://rdrr.io/cran/caret/man/confusionMatrix.html.

Function cv.glmnet. R function for applying Cross-Validation for generalized linear models. URL: https://www.rdocumentation.org/packages/glmnet/versions/4.1-4/topics/cv.glmnet.

Function DecisionTreeClassifier. Python function for implementing a Decision Tree estimator. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

Function GaussianNB. Python function for implementing a Gaussian Naive Bayes estimator. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB.

Function glmnet. R function for fitting a generalized linear model with regularization techniques. URL: https://www.rdocumentation.org/packages/glmnet/versions/4.1-4/topics/glmnet.

Function GridSearchCV. Python function for applying Cross-Validation for estimators. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

Function KNeighborsClassifier. Python function for implementing a KNN estimator. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

Function lda. R function for fitting a Linear Discriminant Analysis model. URL: https://www.rdocumentation.org/packages/MASS/versions/7.3-58.1/topics/lda.

Function multiclass.roc. R function for computing multi-class AUC. URL: https://www.rdocumentation.org/packages/pROC/versions/1.18.0/topics/multiclass.roc.

Function multinom. R function for fitting a single-hidden-layer neural network. URL: https://www.rdocumentation.org/packages/nnet/versions/7.3-18/topics/multinom.

Function MultinomialNB. Python function for implementing a Multinomial Naive Bayes estimator. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB.

Function OneHotEncoder. Python function for encoding categorical features as one-hot numeric arrays. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

Function RandomForestClassifier. Python function for implementing a Random Forest estimator. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Function roc_auc_score. Python function for computing AUC score. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.

Function StandardScaler. Python function for standardizing features. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

Function step. R function for choosing a model by AIC in a stepwise algorithm. URL: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/step.

Function StratifiedKFold. Python function for providing stratified k-folds during Cross-Validation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold.

Function SVC. Python function for implementing a SVM estimator. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.

Function varImp. R function for computing the variable importance for regression and classification models. URL: https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/varImp.

FUT Player Attributes. FIFA 22 players' attributes in FUT modality. By FIFAUTeam. URL: https://fifauteam.com/fifa-22-attributes-guide/.

Goalkeeper. URL: https://www.fifplay.com/encyclopedia/goalkeeper/.

Left Back. URL: https://www.fifplay.com/encyclopedia/left-back/.

Left Forward. URL: https://www.fifplay.com/encyclopedia/left-forward/.

Left Midfielder. URL: https://www.fifplay.com/encyclopedia/left-midfielder/.

Left Wing Back. URL: https://www.fifplay.com/encyclopedia/left-wing-back/.

Left Winger. URL: https://www.fifplay.com/encyclopedia/left-winger/.

Library BeautifulSoup. Python library for extracting data from HTML and XML files. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

Messi Evolution. The Evolution of Lionel Messi's Position and Role. URL: https://www.messivsronaldo.app/articles/messi-position-role-evolution/.

Michael Muller-Mohring. How 'Triple M' Michael Muller-Mohring decides the controversial FIFA 22 player ratings that leaves stars fuming. By Jon Boon, The Sun. URL: https://www.thesun.co.uk/sport/football/7332606/fifa-22-player-ratings-michael-muller-mohring/.

Minimal Cost-Complexity Pruning. URL: https://scikit-learn.org/stable/modules/tree.html#minimal-cost-complexity-pruning.

Position. URL: https://www.fifplay.com/encyclopedia/position/.

Right Back. URL: https://www.fifplay.com/encyclopedia/right-back/.

Right Forward. URL: https://www.fifplay.com/encyclopedia/right-forward/.

Right Midfielder. URL: https://www.fifplay.com/encyclopedia/right-midfielder/.

Right Wing Back. URL: https://www.fifplay.com/encyclopedia/right-wing-back/.

Right Winger. URL: https://www.fifplay.com/encyclopedia/right-winger/.

Rogério Ceni. Stats of the goalkeeper Rogério Ceni. By Wikipedia. URL: https://en.wikipedia.org/wiki/Rog%5C%C3%5C%A9rio_Ceni.

SoFIFA Terms of Service. URL: https://sofifa.com/help/tos.

Striker. URL: https://www.fifplay.com/encyclopedia/striker/.

Terms of Service. What is terms of service (ToS). By Ivy Wigmore, TechTarget. URL: https://www.techtarget.com/whatis/definition/terms-of-service-ToS.

Weak Foot. URL: https://www.fifplay.com/encyclopedia/weak-foot/.

Web Scraping. Step by Step: Web Scraping Using Python BeautifulSoup. By Yalin Yener, Medium. URL: https://medium.com/analytics-vidhya/step-by-step-web-scraping-using-python-36ecb502f8e.

Work Rate. URL: https://www.fifplay.com/encyclopedia/work-rate/.

Ringraziamenti

Innanzitutto, voglio esprimere la mia profonda riconoscenza nei confronti della prof.ssa Annamaria Guolo, che grazie ai concetti appresi durante il suo corso di Data Mining e ai suoi costanti suggerimenti mi ha permesso di poter sviluppare questa tesi. Ho apprezzato sin da subito la sua professionalità, ma ancor di più la gentilezza e la disponibilità che la contraddistinguono.
Un speciale ringraziamento va a papà, mamma e Alessia, i miei punti di riferimento dal giorno zero, che mi hanno insegnato a mantenere la retta via e a perseverare quando il gioco si fa duro. Senza il loro sostegno, questo percorso non sarebbe mai stato possibile. L'ho promessa e l'ho conquistata: questa laurea non è solo il mio obiettivo, ma è anche il loro sogno. Grazie per aver sempre creduto in me. Inoltre, desidero menzionare tutti i miei nonni, zii e cugini, nessuno escluso. So di certo che saranno orgogliosi del raggiungimento di questo obiettivo e del mio percorso, che è più importante di qualsiasi valutazione finale.
Un immenso grazie va ad Elena, la mia spalla destra e la mia certezza, che ha sempre creduto in me e che da poco ha concluso meravigliosamente il suo percorso universitario. Grazie per portare gioia nella mia vita, di riporre fiducia in me e per avermi compreso e sostenuto nei momenti di buio e smarrimento. Per entrambi si apre ora un nuovo capitolo della vita che prevede un percorso in salita e ricco di imprevisti. In ogni modo, sono consapevole che questo percorso sarà molto più piacevole avendo accanto una persona speciale come lei. Ringrazio inoltre i suoi genitori, Luigino e Marinella, che mi hanno sempre ben accolto e sostenuto.
Desidero ringraziare tutti i miei amici più cari con i quali ho condiviso momenti di riflessione e vissuto momenti di goliardia pura. So di non essere stato molto presente in quest'ultimo periodo, ma loro sanno quanto sia stato focalizzato nel raggiungere questo obiettivo. Sono sicuro che da qui in avanti ci saranno molte opportunità per vivere insieme altre giornate e serate memorabili. Inoltre, voglio menzionare gli amici del calcetto, con i quali da anni si organizzano i sentitissimi calcetti settimanali, la nostra valvola di sfogo preferita per condividere la nostra grande passione per il calcio.
Ringrazio tutti gli amici che ho conosciuto all'università e con i quali ho mantenuto un ottimo rapporto a suon di sushi. In particolare, desidero menzionare Federico, Massimo e

Simone per aver condiviso gioie e dolori durante il nostro percorso universitario. Sarò sempre grato a loro per l'amicizia instaurata e il supporto sia morale che tecnico. Auguro il meglio per la loro carriera professionale, e chissà se un giorno a lavoro condivideremo la stessa scrivania.
Con questa tesi si conclude un capitolo della mia vita molto importante, un percorso durato la bellezza di 1971 giorni. Ringrazio l'Università di Padova per avermi dato l'opportunità di apprendere conoscenze teoriche e pratiche che mi serviranno per entrare nel mondo del lavoro. Infine, voglio ringraziare me stesso. Nonostante le iniziali defaillance e le difficoltà legate al periodo pandemico, ho combattuto per ritrovare le giuste motivazioni nel proseguire con gli studi. Auspico di affrontare con la stessa ferocia e determinazione le prossime sfide che il futurò riserverà, sia in ambito personale che professionale.

Padova, Febbraio 2023 Alberto Gobbo