UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Final Dissertation

Dimensionality Reduction and Latent Space Modeling of Soccer Players' Tactical Profiles

Internal Thesis Supervisor

Prof. Marco Zanetti

External Thesis Supervisor

Carlo Bertelli

Candidate

Marco Lorenzetti

Academic Year 2023/2024

Ai miei genitori.

Abstract

Until a few years ago, technical decisions within soccer clubs were based exclusively on qualitative and subjective assessments, relying on the experience of the coaching staff. Nowadays, many clubs have dedicated teams for Data Analysis, which allows them to enhance the efficiency of market movements by objectively analyzing a large dataset of players through precise metrics.
My thesis fits perfectly into this context, aiming to analyze a significant amount of data on individual players to create a tool that aids in player evaluation and describes their playing style. This analysis was conducted alongside my internship at Wolico srl, a recently established start-up founded by my supervisor, Carlo Bertelli. The company has access to a vast amount of data sourced directly from major platforms and has a strong team of Data Scientists who welcomed me into their dynamic work environment.
The first part of this thesis explains how I processed various datasets to extract features for each player. Next, I detail the step-by-step process used to generate heatmaps, which are considered crucial for capturing a player's playing style. Following this introductory phase, I describe the models used to derive numerical vectors from the heatmaps. In this thesis I will refer to these vectors as *play-vectors*. These are the primary tools for analyzing player similarities, as each vector is assumed to summarize a player's style of play.
In my work, I employed two distinct approaches to derive vectors for each player, ranging from a basic factorization method to a technique based on Deep Learning. In particular the two approaches involve *Non-Negative Matrix Factorisation* and *Variational AutoEncoders*. In the final section, I present the results obtained with the different models, highlighting the efficiency of play-vectors in players' characterization, and attempting to interpret the patterns found among the playing styles, obtained by a clustering technique.

Contents

Chapter 1

Introduction

Until a few years ago, soccer was all about natural talent and intuition. Players relied on their skills, while coaches and their teams focused on blending these individual talents into a well organized tactical system.

But this started to change with the rise of computer technology and statistical analysis. The big shift came with the "Moneyball" revolution in the early 2000s [1]. This approach, firstly used by Billy Beane, the former manager of the Oakland Athletics baseball team, involved using stats to find undervalued players who could make a difference. Billy and his team of data scientists, used numbers, data, and player histories to predict performance and make objective decisions about which players to sign and how to use them. At that point soccer clubs took notice and start sawing the potential in using data to find ideal players for specific tactical task at a lower cost.

In recent times, data analysis became even more sophisticated and today it's a crucial part of soccer. Especially thanks to the huge technological advancement over the last 20 years, which has made possible the development of new data collection tools, such as GPS trackers and computer vision algorithms. Nowadays from youth academies to the biggest clubs in the world, data guides decisions, shapes strategies, and boosts performance. Data scientists and statisticians work closely with coaches, turning complex data into clear insights that can influence how teams play and make decisions. One of the most interesting challenges in the world of Data Science applied to soccer is identifying meaningful player's characteristics, avoiding simple frequentist metrics like the number of goals or pass accuracy percentage. Recently, with the exponential growth of available data, more accurate metrics have emerged, aiming to capture even more specific characteristics. One example is Expected Goals (xG), which estimates the likelihood of a shot turning into a goal based on factors like shot position, shot type, and goalkeeper position.

A recourring concept when discussing soccer is a player's style of play. For example, even if Mbappe and Haaland are both great forward players, each one approaches to the game in a very different way. In this context, the goal of my thesis work is to characterize the player's playing style in a data-driven manner, based on analyzing event match data, taken from a public dataset of spatio-temporal match events that cover the entire 2017/18 season of the five major European leagues [2]. Specifically the goal is to summarize the playing style in a fixed-length player vector (which from now on i will call it *play-vector*). From a practical point of view, characterizing playing style if fundamental for three different reasons:

- **Scouting**: Football clubs can better evaluate possible sales and purchases during transfer windows, finding players with specific qualities at a lower price.
- **Monitoring players**: A coach who demands a certain style of play from one of his players

can inspect the evolution of his behavior on the field, and once he matches the expectations, then the coach can monitor that the play-vector remains stable and unchanged in time.

- **Match preparation**: The coach prepares each match basing the strategy on the opponent. If the playing styles of the opponents are available, it's possible to use play-vectors, and dedicated algorithm, to analyze which formations are most effective at breaking through the opposing lines and reaching the goal.

Characterizing a player's style of play from event stream data can be particularly challenging because, unlike traditional statistics, we need to consider the spatial component of every specific action that happens on the field. The solution chosen in my work is to collect each action into a specific 2D histogram, one for each specific type of event choosen. Each element in the histogram represents the frequency of a certain action along the x and y coordinates of the field. These frequencies are then smoothed to create spatial consistency along the coordinates, generating *heatmaps*: the actual input data for the models used. Then i needed to reduce the dimensionality of the heatmaps. In order to do so i approached in two different ways. In the first approach, inspired by the previous work of Decroos and Davis [3], I applied **NMF** (Non-Negative Matrix Factorisation) separately for each type of action. The second approach is based on Deep Learning: I used the heatmaps to train a **Variational AutoEncoder**, obtaining the desired play-vectors from the latent space.

The thesis is divided as follows. In the present Chapter a general overview of the work is presented. Additionally, a section is included that provides an in-depth discussion on the issues present when applying data analysis in high-dimensional spaces, commonly known as the Curse of Dimensionality, as well as a second section presenting related works. In Ch.4 how data are processed in order to generate heatmaps is provided. In Ch.2 and Ch.3 respectively offers a detailed theoretical description of Non-Negative Matrix Factorisation and Variational AutoEncoders; then, in Ch.5 the specific models used in my work are presented. In Ch.6, the analysis on play-vectors obtained are presented, exposing their effectiveness in reflecting playing styles by a *Player Retrieval Analysis*, and exploring the possibility of using the play-vectors to identify sub-patterns among players with a DBSCAN algorithm. Lastly I expose the conclusions on my thesis work.

1.1 Curse of Dimensionality

Dimensionality reduction in data analysis is a concept that is both fundamental and complex. Its use is almost always required, not only to handle high-dimensional data but also to avoid hindering factors such as redundancy and noise. Heatmaps are powerful tools for visually representing the spatial distribution of events, but at the same time, they generate a large amount of data. In fact, each position on the field can be seen as a dimension within the matrix, and considering that the field can be divided into hundreds or thousands of elements, it is clear that representing a player's playing style, even for just one type of action, can include thousands of variables. To effectively manage and analyze this information, it is necessary to find a method that allows us to reduce the large number of variables generated by the heatmaps.

The need to apply dimensionality reduction in my work comes from the high number of variables that the heatmaps add to the model: each matrix is made up of 10404 elements. Then it is easy to see that analyzing the data and finding patterns among them becomes difficult with such a large number of variables. This effect is commonly referred in Machine learning as the 'Curse of Dimensionality' [4], a term coined by Richard E. Bellman to describe the problems related to the drastic reduction in density when a certain number (especially if limited) of data are described by a large number of variables.

The main problems that can be encountered are:

- **Data Sparsity**: when a dataset is described by a large number of dimensions, it is likely that there is a lot of empty space between the datapoints, especially when there is no main component among the dimensions. This distance between points hinders clustering analysis, making it difficult for algorithms to find patterns within the data.

- **Overfitting**: setting up a Machine Learning model with high dimensionality risks making it overly complex, causing it to fit the noise rather than the actual trend.

- **Computational Time**: with an increasing number of variables, more computations are obviously required. As originally understood by Bellman, if we need to approximately optimize a function in d variables and we only know that it is Lipschitz continuous, then on the order of $\frac{1}{\varepsilon^d}$ evaluations are needed to find an approximate minimizer with small error ε [5].

To deal with the high-dimensionality of a dataset many tecniques can be used. A first important classification regarding dimensionality reduction methods can be made:

1. **Features Selection**: this refers to retaining the optimal features and removing the irrilevant ones. It is surely the easiest way to reduce di dimensionality of data. Part of this category are filter methods [6], information gain [7] and correlation coefficient [8].

2. **Features Extraction**: this process involves identifying and extracting new relevant features from the original ones. Specifically, it tries to develop a transformation of the input space onto a low-dimensional subspace that preserves most of the relevant information. Part of this category are well-known tecniques like Principal Component Analysis (PCA) [9], Linear Discriminant Analysis (LDA) [10], Kernel PCA or Deep Learning tecniques such as Autoencoders [11].

The choice between feature selection and feature extraction depends on the specific problem and data, and in some cases, a combination of both techniques might be used for optimal results. However, feature extraction techniques find many more applications because they often reveal deeper structures and relationships in the data, improving the ability to model and interpret complex phenomena. Here is a quick overview of the techniques previously listed.

Principal Component Analysis (PCA)
: PCA applies a linear transformation to the data, computing for each sample new coordinates in order to maximize variance. After applying the transformation to the original dimensions, the covariance matrix is calculated, and its eigenvalues indicate the variance explained by each component. To reduce the number of dimensions, components are selected based on their ability to explain the total variance, choosing those with higher explained variance so that their sum exceeds a predefined threshold.

Linear Discriminant Analysis (LDA)
: LDA looks for the directions in the dataset that maximize the separation between different classes. It calculates the discriminant functions that maximize the ratio of between-class variance to within-class variance.

Autoencoders
: Autoencoders are specific types of neural network made up of two main parts: the encoder and the decoder. The encoder takes high-dimensional input (with many nodes) and, by reducing the complexity through hidden layers, translates it into an output with fewer nodes. The decoder has the opposite job, aiming to reconstruct the original input from the encoder. The low-dimensional space in the middle of this 'bottleneck' structure is called the latent space, and after training, this space represents the new dimensionality of the data

1.2 Related Works

Hyeonah Cho et al. [12] proposed "Pass2Vec", a descriptor vector that can characterize each players' passing style by training a Convolutional Autoencoder with the pass density heatmaps of each player. Decroos and Davies [3] characterized the playing style of soccer players reducing the dimensionality of specific heatmaps using Non-Negative Matrix Factorisation, in such a way that vectors can be human interpretable. Both two researchs was for me a guideline to develop the basis of my work. In my research, in fact, I applied both tecniques in order to extract meaningful and representative playing style vectors for each player in the dataset. However, my work takes into account an higher number of features, trying to lose less information possible. Especially, one of the main differences from Hyeonah Cho et al.'s work is my use of a more advanced algorithm, the Variational Autoencoder, which incorporates a statistical a priori distribution on the data's arrangement in the latent space. These are the only two works I have found that aim to achieve the same goal as mine: condense a soccer player's playing style into a vector.
The work proposed by Campos [13] develops a method to create vector representations of passing actions in football using Graph Convolutional Networks (GCN). This approach considers player positions and pass characteristics to generate embeddings that reflect the context of the actions. Unlike my work, which uses dimensionality reduction techniques to analyze heatmaps and obtain vectors representing playing style, this method focuses on analyzing passing actions and predicting surrounding actions within the game context. The framework aims to enhance the understanding of game actions through a detailed and contextualized representation.

Chapter 2

Non-Negative Matrix Factorization

In this section, I provide an overview of Non-Negative Matrix Factorization (NMF), starting with an introduction to the problem and its mathematical formalism. Next, I briefly introduce the main algorithms used to solve it. The formal steps leading to the updating rules would require numerous detailed steps and prior knowledge. Finally, I expose the applications of NMF in Machine Learning providing an example.

2.1 Problem Definition

The Non-Negative Matrix Factorization (NMF) was initially introduced by Paatero et al. [14] under the name *Positive Matrix Factorization*, and later developed and made famous by Lee and Seung [15]. This technique involves decomposing a matrix V, defined as non-negative, into two matrices W and H, which are also non-negative. The non-negativity is a key element in this technique, setting it apart from other methods like PCA or Vector Quantization [16], which include the use of negative values. These constraint lead NMF to a component-based representation, allowing to express each data input as linear (and mainly positive) combination of principal components.
Let $V = (v_1, v_2, ..., v_n) \in \mathbb{R}_+^{m \times n}$ be the input non-negative matrix. As before mentioned the NMF decompose V as follows:

$$V \approx W \cdot H,$$
$$V_{ij}, W_{ij}, H_{ij} \geq 0, \quad \forall i, j$$

where $W = (w_1, w_2, ..., w_k) \in \mathbb{R}_+^{m \times k}$ and $H = (h_1, h_2, ..., h_n) \in \mathbb{R}_+^{k \times n}$. k is a fixed parameter with the constraint $k < \min(m, n)$. As can be seen from the formula mentioned above, although it is possible to prove that the exact decomposition of V exists, the computation of the two matrices is not computationally trivial at all. Therefore, in most cases, we refer to it as approximation. Then we can formalize the search for the matrices W and H into a minimization problem:

Problem

Given an $m \times n$ non-negative matrix V and an integer $k < \min(m, n)$, solve

$$\min_{W \in \mathbb{R}_+^{m,k},\ H \in \mathbb{R}_+^{k,n}} \|V - W \cdot H\|_F^2 \tag{2.1}$$

where the loss function indicated is the Frobenius distance, which can be seen as the generalization of the Euclidean norm to matrices:

$$\|x\|_F^2 = \sqrt{|x_1|^2 + |x_2|^2}$$

Applied to 2.1, this leads to the following formulation:

$$\min_{W,H} \sqrt{\sum_j^m \sum_i^n [V_{ij} - (WH)_{ij}]^2}. \tag{2.2}$$

To achieve the W and H that minimize the Frobenius distance, different approaches exist. In the following i will exposed briefly two of the most used.

2.1.1 Multiplicative Updating Rule

The first one, introduced by Lee and Seung, is probably the most famous approach due to its simplicity in the implementation, and can be summarized into the following scheme:

Algorithm 1 Multiplicative Updating Rule

1: Initialize W, H with random non-negative values.
2: **repeat**
3: Update W:

$$W_{i\mu} = W_{i\mu} \frac{\left(VH^T\right)_{i\mu}}{\left(WHH^T\right)_{i\mu}},$$

where $\mu = 1, 2, \ldots, k$.
4: Update H:

$$H_{\mu j} = H_{\mu j} \frac{\left(W^T V\right)_{\mu j}}{\left(W^T WH\right)_{\mu i}}.$$

5: **until** $\|V - W \cdot H\|_F^2$ is below a certain threshold.

In the original paper [15], it is proved that under these updating rules, the Frobenius norm is non-increasing. This ensures the validity of the iterative algorithm. However, it is well-known that this method may easily get stuck in local maxima, leading to low computational efficiency. Being a multiplicative iterative process, we need to avoid reducing all matrix values to zero when encountering a zero value in the update rule: it is good practice to add an infinitesimal constant $\varepsilon \sim 10^{-6}$ to every zero value.
The minimization problem is often expressed in terms of **log-likelihood** of V_{ij} as a variable drawn from a Poisson Distribution with scale $(WH)_{ij}$:

$$\min_{W,H} L(W,H) = \min_{W,H} \sum_j^m \sum_i^n V_{ij} \log (WH)_{ij} - (WH)_{ij}.$$

In this approach, the process remains multiplicative, but the updating formulas change, as summarized into the following scheme:

Algorithm 2 Multiplicative Updating Rule (Log-likelihood Approach)

1: Initialize W, H with random non-negative values.
2: **repeat**
3: Update W:

$$W_{i\mu} = W_{i\mu} \frac{\sum_j^m H_{\mu j} \frac{V_{ij}}{(WH)_{ij}}}{\sum_j^m H_{\mu j}},$$

where $\mu = 1, 2, \ldots, k$.
4: Update H:

$$H_{\mu i} = H_{\mu i} \frac{\sum_i^n W_{i\mu} \frac{V_{ij}}{(WH)_{ij}}}{\sum_i^n W_{i\mu}}.$$

5: **until** $\|V - W \cdot H\|_F^2$ is below a certain threshold.

2.1.2 Alternating Least Squares

Another widely used approach, in order to better deal with the issues in the optimization problems, is to fix one matrix to solve a minimization problem only for the other one. This way, we are able to solve the non-convexity of the multivariate plane.

Algorithm 3 Alternating Least Squares (ALS)

1: Initialize W, H with random non-negative values.
2: **repeat**
3: Fix H and update W by solving the Least Squares problem:

$$\min_{W \geq 0} \|V - WH\|_F^2$$

4: Fix W and update H by solving the Least Squares problem:

$$\min_{H \geq 0} \|V - H^T W^T\|_F^2$$

5: **until** Convergence.

Even though this method helps to deal with the non-convexity, this algorithm is very slow: even faster implementations can struggle to match other methods in terms of running time. Several updated versions have been proposed. One of the most famous replaces the exact solution, which deals with the non-negative constraint, with an inexact solution for the Least Squares unconstrained method, projecting at each iteration the matrix obtained into the non-negative space. The procedure can be summarized as follows:

Algorithm 4 Inexact Alternating Least Squares with Non-Negative Projection

1: Initialize W, H with random non-negative values.
2: **repeat**
3: Fix W and update H solving the equation $HW^TW = VW$
4: Project H into the non-negative space: $H = [H]_+$
5: Fix H and update W solving the equation $WH^TH = VH$
6: Project W into the non-negative space: $W = [W]_+$
7: **until** Convergence.

This implementation spends significantly less time to converge but sacrifices the convergence quality and the goodness of the factorization. Nowadays, more sophisticated versions have been

developed that better deal with convexity and convergence time, such as Projected Gradient Descent or Coordinate Descent.

2.2 Machine Learning Applications

NMF, as powerfull unsupervised learning tool, finds application in many fields such as computer vision, audio signal processing, clustering, bioinformatics and many others. As mentioned before, the beauty of NMF tecnique is the possibility to determine the main components that better represent the whole dataset and for each data sample detect the non negative linear combination of the components. From a practical point of view it allows us to find, in a totally unsupervised approach, the main elements that describe our data and provide us a compact representation of the whole dataset. These characteristics make it particularly useful because of its human interpretability.

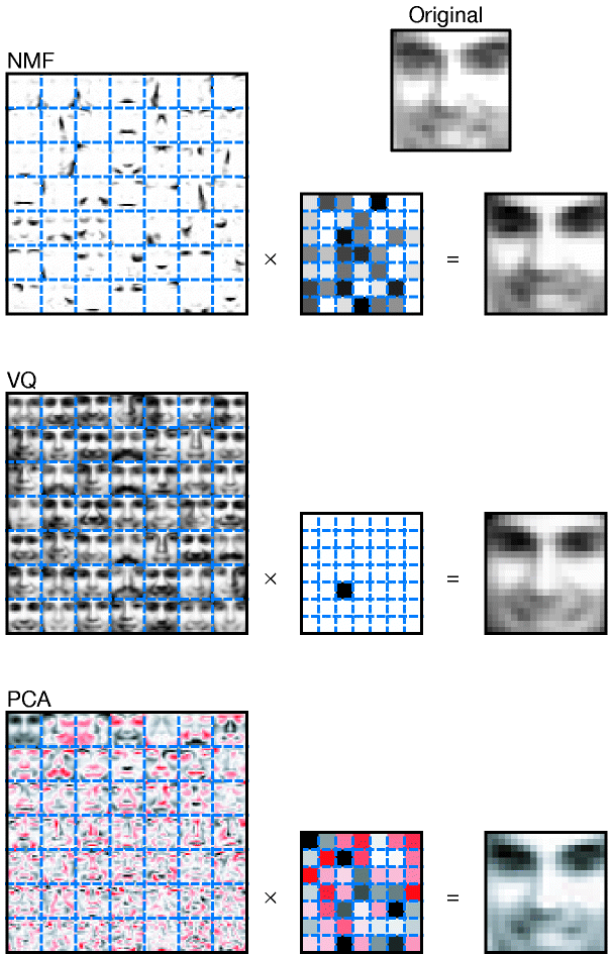


Figure 2.1: Comparison between three learning methods presented in original Lee and Seung paper [15]. Non-Negative Matrix Factorization learning a parts-based representation of faces (at the top), Vector Quantization (in the middle) and Principal Component Analysis (at the bottom) learning holistic representations.

A proper example of the power and of the advantages introduced by NMF if given by Lee and Seung themselves in one of their first publications about the topic. In their work compared NMF, PCA and Vector Quantization. The learning methods were applied to a dataset of facial images, each consisting of 19×19 pixels. Once the flattened the images in $1D$ vectors, they merged this ones all togheter to form the input V matrix. The goal, as explained until now, was to derive the

W transofrmation matrix, which each column represent the basis of the transformention, and the H matrix, which contain the compact representation of flattened images. In figure 2.1 the three mathods based approximated factorizations are shown. At left a grid shows all main components (basis) found by each method; in the center the weights grids for the image recontructions, where the intensity of each weight gets whiter increasing its value; which are shown at right. As can be seen the NMF basis and encodings contain a lot of vanishing weights, which lead to a sparse basis matrix. Without entering in complex explanations we can state that this happens due to the fact that NMF learn as basis different version of different face parts, while the other tecniques seems to learn as basis common features like spiecific edges (in PCA) and specific faces (in VQ). Surely it give a huge merit to the NMF tecnique, in fact is much easier (and much more human interpretable) to categorize faces addressing specific attributes to each face part.

Chapter 3

Variational Autoencoder

Autoencoders are Deep Learning algorithms whose task is to take input data, reduce its dimensionality, and reconstruct the original data as output [17]. The input is mapped to a lower-dimensional space, called latent space, to prevent the algorithm from learning the trivial identity transformation. To achieve this, a classic autoencoder is structured with two symmetric neural networks: encoder and decoder. For semplicity in this section we will consider a linear input, but every consideration can be made for multidimensional input. The encoder translates the original input $x \in \mathbb{R}^D$ into its compact form in the latent space $z \in \mathbb{R}^L$, where $L < D$. Then, the decoder maps z back into $\mathbb{R}^D$, producing an output $\tilde{x}$. A visual representiation is given by figure 3.1. The quality of the output is quantified by a loss function $L(x, \tilde{x})$, where the metric depends on the specific problem.
As mentioned earlier, the task of an autoencoder is to reconstruct the original input, using a technique that involves mapping it into a lower-dimensional space. However, this algorithm has much greater capabilities, and depending on its structure and the chosen error metric, it is possible to train the model to perform more interesting tasks, such as removing noise (Denoising Autoencoders) [18] or generating missing data portions (Inpainting Autoencoders) [19].
There are many advanced types of autoencoders. The ones of interest for my thesis work are the Variational Autoencoders [20], where data is mapped into the latent space while attempting to give statistical consistency to the mapping. This modification not only allows for better separation of data from different categories in the latent space but also provides a distribution such that the *Latent Space-Decoder* pair becomes a true generative algorithm.
To adequately cover this topic, this chapter is divided as follows: in the first section, the Kullback-Leibler Distance is introduced, a key concept for fully understanding the mechanism of Variational Autoencoders. In the following section, the problem addressed by this algorithm is explained, along with the theoretical development that leads to the definition of the metric chosen for the loss function.

3.1 Kullback-Leibler Divergence

The steps needed to fully understand how a Variational Autoencoder works, and particularly the mathematical approach behind the optimization of its parameters, require us to start from basic concepts. Firstly, the concept of Kullback-Leibler divergence and the related preconceptions from information theory [21].

Suppose we have a variable X and a set of possible outputs $x_n \in X$. In information theory, the amount of information gained from a certain output x is calculated as

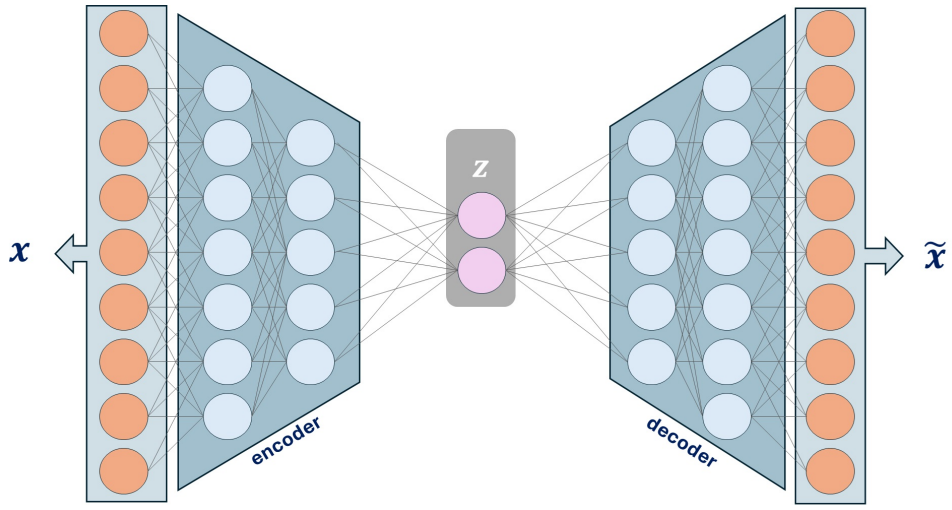


Figure 3.1: Classic Autoencoder Structure. An input x is passed to the encoder, which elaborate it by a sequence of convolutional or linear (as presented in the figure) layers and return an output z whit a reduced dimensionality. Then z is used as input for the decoder, which restores the original dimensionality, trying to reconstruct the data.

$$I(x) = -\log p(x),$$

where $p(x)$ is the probability associated with the event x. Therefore, given a set of events and the corresponding probability distribution, we can compute the expected value of the information gained from an output of X. This quantity is called Shannon Entropy:

$$H(X) = -\sum_{i=1}^{n} p(x) \log p(x).$$

From this definition, we can extend the analysis of the information obtained to the case of two correlated (or not) variables.
We define Joint Entropy as:

$$H(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x,y),$$

where $p(x,y)$ is the joint probability of having both x and y as outputs, which by definition can be calculated as

$$p(x,y) = p(x|y)p(y) = p(y|x)p(x),$$

where $p(x|y)$ is the conditional probability of having output x from sampling X after having obtained y from sampling Y.
Through this definition, it is possible to formulate Conditional Entropy:

$$H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x|y).$$

This is particularly important as it represents the average amount of uncertainty that remains after first sampling from another variable.

To move to the next step, given the introduction just made about the fundamental preconceptions of information theory, let's define a new statistical scenario. Let's take a variable X and a set of possible events $x_n \in X$. Consider two different distributions of X: $p(x)$ and $q(x)$, whose respective entropies $H(p)$ and $H(q)$ represent the average uncertainty when sampling from these two distributions. If we first sample from q, the average uncertainty will be given by the conditional entropy $H(p|q)$. It is logical to think that if p and q are identical distributions, then sampling from p, after first sampling from q, will be redundant since it would maintain the same uncertainty:

$$H(p|q) = H(p) = H(q), \quad \text{if } p(x) = q(x)\ \forall x \in X.$$

From this observation we define the Kullback-Leibler divergence as follows:

$$D_{KL}\left[p||q\right] = H(p|q) - H(p), \tag{3.1}$$

which can be interpreted as the difference between the uncertainty gained by sampling from $p(x)$ when first sampling, and not, from $q(x)$. Another way to interpret D_{KL}, which will be useful later, is as a quantity that measures the inefficiency of $q(x)$ when sampling from X, assuming its true distribution is $p(x)$.
Through a simple mathematical manipulation, we can reformulate 3.1 as

$$D_{KL}\left[p||q\right] = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} = \mathbb{E}_{p(x)}\left[\log \frac{p(x)}{q(x)}\right].$$

It is important to note that this quantity is non-symmetric ($D_{KL}[p||q] \neq D_{KL}[q||p]$), only admits non-negative values ($D_{KL} \geq 0$), and although it is commonly understood as a distance, it is not such in the traditional geometric sense, as it does not satisfy the triangle inequality and is not symmetric.

3.2 Problem Definition

The structure of a VAE, like classical autoencoders, reduces the dimensionality of the input data by projecting it onto the latent space, but the approach behind the VAE imposes that the distribution of the data projected in the latent space follows a certain prior $p(z)$, which best reflects the actual distribution of the data described by $p(x)$ [22]. To achieve this, as we will see better at the end of the section, a stochastic component is introduced in the mapping of x onto the latent space. A visual representation of the model structure, which will be more clrear at the end of the chapter, is presented in figure 3.2. Our goal is to train encoder and decoder to best represent the conditional probability distribution $p(z|x)$, which is related to the prior by Bayes' theorem:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int_{\mathbb{R}^L} p(x|z)p(z)dz}.$$

The integral at the denominator is calculated over all the dimensions of the latent space and it is not easily to solve analytically. Since we cannot use a Bayesian approach, we can instead use a variational approach. In Variational Inference is defined a conditional probability distribution $q(z|x)$ which best approximates $p(z|x)$. To make these distributions as similar as possible, we use the Kullback-Leibler Distance, as we saw in the previous section. The problem can thus be defined as:

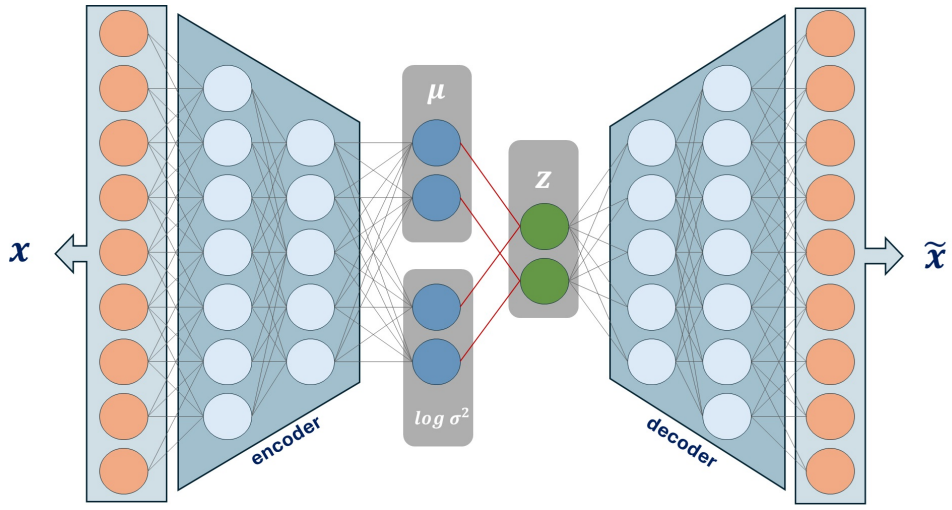


Figure 3.2: Variational Autoencoder structure. An input x is passed to the encoder, which elaborate it by a sequence of convolutional or linear (as presented in the figure) layers and return an output two vectors μ and $\log \sigma^2$, which respectively represent the mean and the log-variance of the desired distribution. Sampling from this distribution we are able to compute z, used as input for the decoder, which restores the original dimensionality, trying to reconstruct the data.

$$\min_{q \in Q} D_{KL}[q(z|x)||p(z|x)],$$

where Q represents a set of possible probability distributions, called the Variational Family. As we have just observed, it is not possible to deal with $p(z|x)$ directly, so we need to manipulate this definition:

$$\begin{aligned} D_{KL}[q(z|x)||p(z|x)] &= \mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z|x)}\right] = \mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(x|z)p(z)} + \log p(x)\right] \\ &= \mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right] - \mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] + \log p(x) \end{aligned}$$

Rearranging, we obtain:

$$\log p(x) = -\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right] + \mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] + D_{KL}[q(z|x)||p(z|x)], \tag{3.2}$$

where $\log p(x)$ represents the likelihood of the reconstructed data, a quantity we aim to maximize (analogous to minimizing the Kullback-Leibler Distance). We notice that, since $D_{KL} \geq 0$, the first two terms in 3.2 act as a *lower bound* to $\log p(x)$. Therefore, these are summarized in a single term called ***Evidence Lower Bound*** [22]:

$$\text{ELBO} = -\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right] + \mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] \leq \log p(x). \tag{3.3}$$

In aiming to maximize $\log p(x)$, we can translate the problem into the maximization of ELBO. However, we can observe in 3.3 that the first term can be traced back to a Kullback-Leibler divergence:

$$\text{ELBO} = \mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] - D_{KL}\left[q(z|x)||p(z)\right] \tag{3.4}$$

In practice, we need to quantify the performance of an algorithm through the estimation of an error, in order to then update the weights via backpropagation. Therefore, the error calculated by the VAE is defined as the opposite of ELBO, a quantity directly proportional to the goodness of the algorithm:

$$\mathcal{L}_{VAE} = -\text{ELBO} = \underbrace{-\mathbb{E}_{q(z|x)}\left[\log p(x|z)\right]}_{A} + \underbrace{D_{KL}\left[q(z|x)||p(z)\right]}_{B},$$

A represents the quality of the output reconstruction of the input data, pushing the algorithm to focus on increasingly accurate reconstructions; in practice, this term is replaced by metrics such as the Mean Squared Error (MSE) or the Binary CrossEntropy (BCE), which will be better discussed in section 5.2.1.

B represents the distance between the actual distribution of the data in the latent space and the imposed one $p(z)$. Usually, both are represented by normal distributions where

$$p(z) = \mathcal{N}(z|0, I), \tag{3.5}$$
$$q(z|x) = \mathcal{N}(z|\mu, \sigma^2) \tag{3.6}$$

which relative logarithms are

$$\log p(z) = -\log\left(\sqrt{2\pi}\right) - \frac{z^2}{2} \tag{3.7}$$
$$\log q(z|x) = -\log\left(\sigma\sqrt{2\pi}\right) - \frac{(z-\mu)^2}{2\sigma^2} \tag{3.8}$$

Developing $D_{KL}\left[q(z|x)||p(z)\right]$ we get:

$$D_{KL}\left[q(z|x)||p(z)\right] = \mathbb{E}_{q(z|x)}\left[\log\frac{q(z|x)}{p(z)}\right] = \mathbb{E}_{q(z|x)}\left[\log q(z|x) - \log p(z)\right]. \tag{3.9}$$

Now, substituting 3.7 and 3.8 in 3.9 we obtain:

$$D_{KL}\left[q(z|x)||p(z)\right] = \mathbb{E}_{q(z|x)}\left[-\log\left(\sigma\sqrt{2\pi}\right) + \log\left(\sqrt{2\pi}\right) - \frac{(z-\mu)^2}{2\sigma^2} + \frac{z^2}{2}\right].$$

After some easy manipulations we get to the following form:

$$D_{KL}\left[q(z|x)||p(z)\right] = -\frac{1}{2}\left[1 + \log\sigma^2 - \mu^2 - \sigma^2\right]. \tag{3.10}$$

We can therefore observe from 3.10 that the Kullback-Leibler Divergence is calculated based on only two parameters, which are the same ones needed to compute z: μ and σ. For convenience, the procedure is parameterized by considering $\log\sigma^2$ instead of σ. Consequently, the structure of the VAE is such that the input x is reduced to two vectors of the same dimensionality as the latent space, corresponding to μ and $\log\sigma^2$, which are then used to compute the elements of z. A graphical representation is presented in figure 3.2.

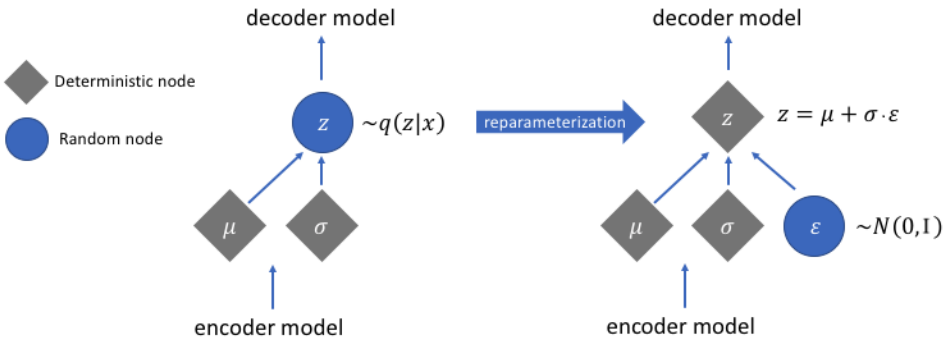


Figure 3.3: Reparametrization Trick scheme, it allows the differentiable sampling of z, separing the noise ε from the deterministic μ and σ.

3.2.1 Reparametrization Trick

As explained before, our goal is to sample z from a $q(z|x)$ that is as similar as possible to $p(z|x)$. The sampling of z is characterized by a stochastic component, which makes it impossible to optimize the VAE parameters using gradients.
The Reparametrization Trick is a strategy, represented visually in figure 3.3, used to avoid this problem [23]. This does not involve sampling z directly from a normal distribution $\mathcal{N}(z|\mu, \sigma^2)$, but parametrizing the sampling using Gaussian noise $\varepsilon \sim \mathcal{N}(0, I)$:

$$z = \mu + \sigma \cdot \varepsilon.$$

Now, the sampling of z becomes differentiable with respect to μ and σ, making backpropagation possible.

Chapter 4

Data Processing

In this section, I describe how, starting from a dataset of events, I developed a set of heatmaps divided by type of action. These are used later as input data to get play-vectors using dimensionality reduction.

4.1 Event Dataset

The dataset used is part of an open collection of soccer logs covering the 2017/18 season in the top five European leagues (Premier League, La Liga, Serie A, Bundesliga, and Ligue 1) and two international cups between national teams (World Cup 2018 and European Cup 2016). According to the authors, "it is the largest collection of soccer logs ever released to the public" (2019, [2]). The data described in the original paper were collected and provided by WyScout [24], a leading company in the soccer data industry. The data collection process is carried out by expert video analysts (the operators), who are properly trained to complete this task using the "tagger," a dedicated software. For each match, three operators (four for live data) handle the data collection. Afterward, a quality control process takes place: first, an algorithm detects major errors by comparing the operators' logs to ensure consistency, and then a more detailed manual review is done by other operators. For my thesis work, I decided to limit the data to the five European leagues present in the original dataset. The table 4.1 includes a list of the selected competitions, and for each, the number of matches, events, and players is reported.

Competition	**#matches**	**#events**	**#players**
Spain	380	628,659	619
England	380	643,150	603
Italy	380	647,372	686
Germany	306	519,407	537
France	380	632,807	629
Total	**1826**	**3,071,395**	**3,074**

Table 4.1: Number of matches, events, and players for each competition.

Each row in the dataset represents a specific event from over 3 million available. Each event is described by multiple fields, which we can summarize into the following 3 categories of information: player fields (including name, role, and team), match fields (which includes information like date and first or second half), and action fields. In the action-related ones, what matters most to us are the type (or sub-type) of action and the field coordinates. These coordinates are given as a percentage from the start and end of the field, so we can collect data consistently regardless

of the stadium size. The first goal we need to set at this stage is to choose the types of actions that best describe a player's playing style, and the following section has the objective to figure it out.

4.1.1 Action Types Selection

The table 4.2 lists all the types of events in the dataset and their sub-types, for a total of 10 types and 36 sub-types. To describe playing style, it's not necessary to consider all the features. Many of them are not truly defining or are redundant (for example, the sub-type "Touches", which in a typical game action usually comes before a more important action like *Pass* or *Shot*). In agreement with Decroos and Davis [3], I considered it appropriate to assume that the relevant actions are those with the ball in play, excluding all events related to set pieces: like penalties, free kicks, or corners. This is because set pieces are often taken by specialists with specific technical skills, and the action itself is disconnected from the overall game flow. To provide an example, Aleksandar Kolarov, whose even if is a left-back defender, scored a total of 10 goals from free kicks while playing for Manchester City. Even though it might seem like this feature defines a player, it's important to understand that this study focuses on players' behavior with the ball in play, and features like set pieces certainly deserve a deeper analysis, but one different from the one presented in this thesis.

Event type	**Sub-types**
Duel	Air duel, Ground attacking duel, Ground defending duel, Ground loose ball duel
Foul	Foul, Hand foul, Late card foul, Out of game foul, Protest, Simulation, Time lost foul, Violent Foul
Free Kick	Corner, Free Kick, Free kick cross, Free kick shot, Goal kick, Penalty, Throw in
Goalkeeper leaving line	Goalkeeper leaving line
Interruption	Ball out of the field, Whistle
Offside	—
Others on the ball	Acceleration, Clearance, Touch
Pass	Cross, Hand pass, Head pass, High pass, Launch, Simple pass, Smart pass
Save attempt	Reflexes, Save attempt
Shot	Shot

Table 4.2: List of events' types and their corresponding sub-types.

Another criterion applied is to focus only on "on the ball" events: that's why features like *Foul* type was excluded from this analysis.
After applying the above criteria, the remaining types was *Duels*, *Passes*, and *Shots*. However, we have to make three important observations. The first concerns passing style. Each type of pass has its function, and the listed sub-types highlight this well. I found it too detailed to break down this category into all the specific forms, so I simplified it into two simple categories: long passes and short passes. Long passes include *Cross* and *Launch* (from now on, I will refer to this feature as ***Cross***), while short passes include all other sub-types (from now on, I will refer to this feature as ***Pass***). The second observation concern the information contained in the Pass event type. In fact the pass it self can't just be described by the starting location, but one of the most interesting aspects is where the player is passing the ball. This feature tells a lot about the player connection with the rest of the team. That's why i chose to split the Pass feature in two different ones: ***Pass-start*** and ***Pass-End***. The third and final observation concerns duels, as this type can provide information about a player's attacking and defensive style: by

selecting *Ground attacking duel* and *Ground defending duel*, respectively. The remaining two do not clearly express an attacking or defensive condition, and since they could be either, they were left out. From now on, I will refer to these two features as ***Dribbling*** and ***Tackle***, respectively. In the work by Decroos and Davis, Tackles were excluded because it was believed that defense is, by definition, positional and relies little on tackles made during the game. While I agree with the positional (and tactical) nature of defensive behavior, I believe it is a missed opportunity to exclude a feature that still serves as an indicator of a defensive role during non-possession phases, like tackles. Therefore, they are included in this study. No processing for ***Shot*** has been made.
The following six features remain: Pass-Start, Pass-End, Cross, Shot, Dribbling, and Tackle. At this stage, removing all the other event types remain 2642700 instances in the dataset ($88,66\%$ of the original one).

4.1.2 Player's Playing Time Filtering

Players who had low playing time during the 2017/18 season were excluded from the dataset. This was done to avoid noise, meaning the presence of players whose actions were not frequent enough to effectively define their playing style. To do this, was used a second dataset consistent with the previous one, but which provides more detailed information about each match, specifically the line-up for each game and the related substitutions [25]. Based on this information, a Python script was developed to calculate the playing time of each player in the dataset for the season. In order to verify the goodness of the minutes derived, i checked the actual playing time of some players on the Fbref website [26] and compared it with the playing time that i computed. In the table 4.3 a comparison of Fbref minutes and mine ones, for the starting line up formation of S.S. Lazio, shows the validity of the minutes obtained.
After i chose a threshold to cut out the players with low playing time. Many papers use a threshold of 900 minutes of playtime, but this time threshold is quite arbitrary.
As explained in the next chapter, I will compare the playing frequencies between the first and second halves for each player. Therefore, it is necessary that each selected player has played a significant amount of time in both halves. Consequently, I calculated the minutes played by each player in both the first and second halves. Then, I computed the 25^{th} percentile for each half ($P_{25}^{1H} = 225$, $P_{25}^{2H} = 255$) and excluded players whose playing time in either half did not reach the respective percentiles. In this way, 1787 players remained.

Player Name	**Minutes (Fbref)**	**Minutes (My Code)**
Ciro Immobile	2683	2741
Ştefan Daniel Radu	2588	2609
Stefan de Vrij	3034	3039
Alessandro Murgia	777	851
Luiz Felipe Ramos Marchi	1168	1164
Thomas Strakosha	3420	3420
Senad Lulić	2426	2447
Felipe Anderson Pereira Gomes	1162	1152
Sergej Milinković-Savić	2844	2848
Adam Marušić	2523	2551
Lucas Pezzini Leiva	2882	2894

Table 4.3: Comparison of S.S. Lazio's starting line up players minutes of effective play: Fbref vs. My Python script calculations.

4.2 Heatmaps Generation

In this section, is described the process that, starting from the dataset filtered according to the criteria listed above, allowed me to obtain a set of 1787 heatmaps for each type of action. In this chapter, we will use the following notation:

Notation Legend

f	:	feature/action type.
p	:	p-th player.
$\mathbf{X}(p, f)$	:	Grid/Heatmap of p-th player for f feature type.
$X_{ij}(p, f)$	:	Element of the matrix $\mathbf{X}(p, t)$ at row i and column j.

According to the original dataset paper, each event reports the coordinates where it occurred. Both coordinates, horizontal and vertical, are expressed as integers ranging from 0 to 100: a percentage representation of the actual field dimensions. By the way was observed that for both dimensions, the maximum value is actually 101, assuming that 0 and 101 refer to areas outside the field (no specific explanations was found on the original paper). Then a grid $\mathbf{X}(p, f) \in \mathbb{R}^{l \times l}$, where the size $l = 102$, is created for each player, and the value of $X_{ij}(p, f)$ is incremented by 1 whenever an event occurs at the percentage coordinates (i, j) on the playing field. At the end of the collection process, we obtain an estimate of the spatial frequency for each event f and each player p, resulting in a total of $10,962$ grids (1787 grids $\cdot$ 6 action types). From now on, for simplicity, I will omit the subscripts (p, f), referring to each grid as $\mathbf{X}$.
The next step was to normalize each grid based on the actual playing time of each player: the goal is to represent game frequencies relative to the duration of a standard soccer match, which is 90 minutes. Therefore, each grid $\mathbf{F}$ was processed as follows:

$$\mathbf{X}' \longrightarrow X'_{ij} = \frac{X_{ij}}{T_p} \cdot 90, \qquad \mathbf{X}' \in \mathbb{R}^{l \times l} \tag{4.1}$$

where T_p represents the actual playing time of the p-th player, computed previously.
At this point, the heatmaps only have representativeness at specific points, leaving much of the playing field empty. This is not ideal because we need a consistent statistical representation across all spatial coordinates. In fact, if a point on the grid shows a certain frequency of play, it is reasonable to assume that, in reality, it is statistically likely that similar frequencies occur in nearby surrounding coordinates. Therefore, each grid is processed with a **Gaussian filter**, which has the function to smooth the discrete frequencies in a more spatial-continuos form.
In 2D Gaussian filtering, the process involves convolving the input grid $\mathbf{H}'$ with a Gaussian kernel [27]. The Gaussian kernel is defined as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \tag{4.2}$$

where (x, y) are the coordinates in the kernel, and σ is the standard deviation of the Gaussian distribution. The convolution of the grid $\mathbf{X}'$ with the Gaussian kernel produces the filtered grid $\mathbf{X}''$:

$$X''_{ij} = \sum_{k=-K}^{K} \sum_{u=-K}^{U} X'_{(i+k)(j+u)} \cdot G(k, u), \qquad \mathbf{X}'' \in \mathbb{R}^{l \times l} \tag{4.3}$$

where K and U are the dimensions of the kernel, and the summation covers all the pixels in the neighborhood around (i, j). To implement the 2D Gaussian Filtering the `scipy.gaussian_filter()`

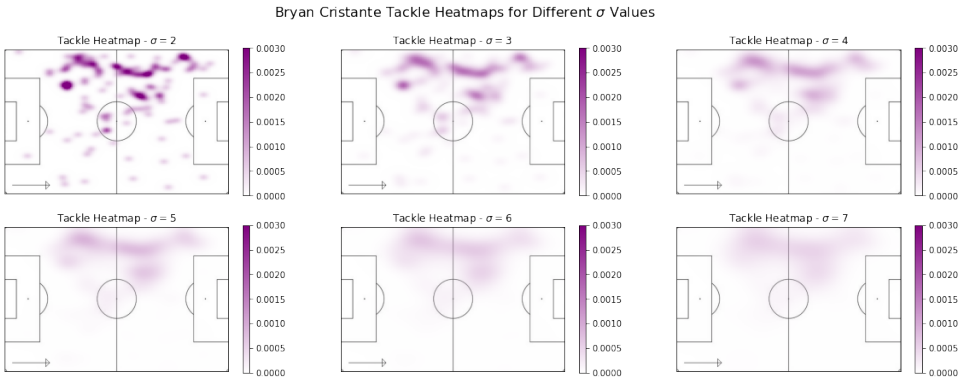


Figure 4.1: Different Blurring on Brayan Cristante (Midfielder, AS Roma) Tackle Heatmaps in function of Gaussian Filter Hyperparameter σ.

[28] method was used. This into account the σ parameter; then K and U are equally set as $K = U = \texttt{truncate} \cdot \sigma$, where the hyperparameter `truncate` is set to 4 by default, obtaining a squared kernel of shape $(4\sigma, 4\sigma)$. Different values for σ has been tried. In figure 4.1 are shown some gaussian filtering trials for different values of σ: it can be observed that to ensure good spatial generalization while simultaneously avoiding excessive spreading of intensity, $\sigma = 5$ represent a valid choice. Smoothing through Gaussian filtering significantly reduced the intensities of each element in the grid. Since the values of the heatmaps are used as input data to train the models, it is necessary to apply one final transformation to stretch the value distribution into a wider range [29]. There are several methods to achieve this transformation, and it's important to consider different aspects. First of all, the transformation must be linear. In fact, the values

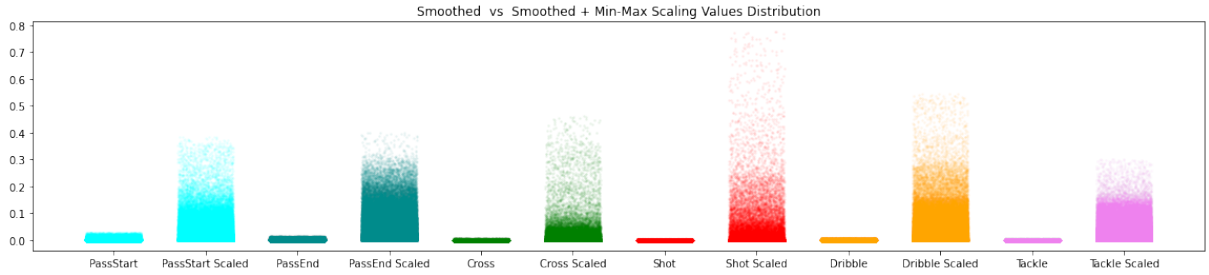


Figure 4.2: Min-Max Scaling effect. For each feature (x-axis) are compared the intensities (y-axis) of all heatmap values, before (left) and after (right) Min-Max scaling. Can be clearly seen how the scaled values cover a wider range than the not scaled ones.

contained in each element of the heatmaps represent the frequency of a specific action on the field, and it is crucial that the frequencies within the same heatmap remain spatially consistent with each other. For example, applying a quadratic transformation could cause an imbalance in the frequencies, making a player with even distribution across the field appear as if they are focused on just one area. It is also important to consider that the heatmaps are used to compare playing styles between players, so the ratio between the frequencies of two different players must remain unchanged after the transformation. Finally, many of the algorithms used to extract player vectors require positive values, so it is essential that the transformation returns $X_{ij} > 0$. For the reasons mentioned above, Min-Max Scaling was chosen, a transformation that normalizes the input values into a range between 0 and 1, as follows:

$$X'''_{ij} = \frac{X''_{ij} - X''_{min}}{X''_{max} - X''_{min}}, \quad \mathbf{X}''' \in \mathbb{R}^{l \times l} \tag{4.4}$$

where $\mathbf{X''_{min}}$ and $\mathbf{X''_{max}}$ refer to the minimum and maximum intensity, respectively, within the entire set of grids of a given type f. In figure 4.2 a comparison of values distribution before (just smoothed) and after (smoothed + scaled) the Min-Max Scaling.
Finally, at the end of this processing stage, we got the finished 6-types heatmaps for all 1787 players. In figure 4.3 and 4.4 a visual recap of the grids' processing.

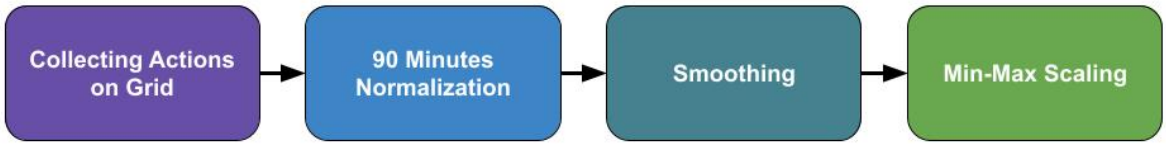


Figure 4.3: Heatmaps Generation Scheme

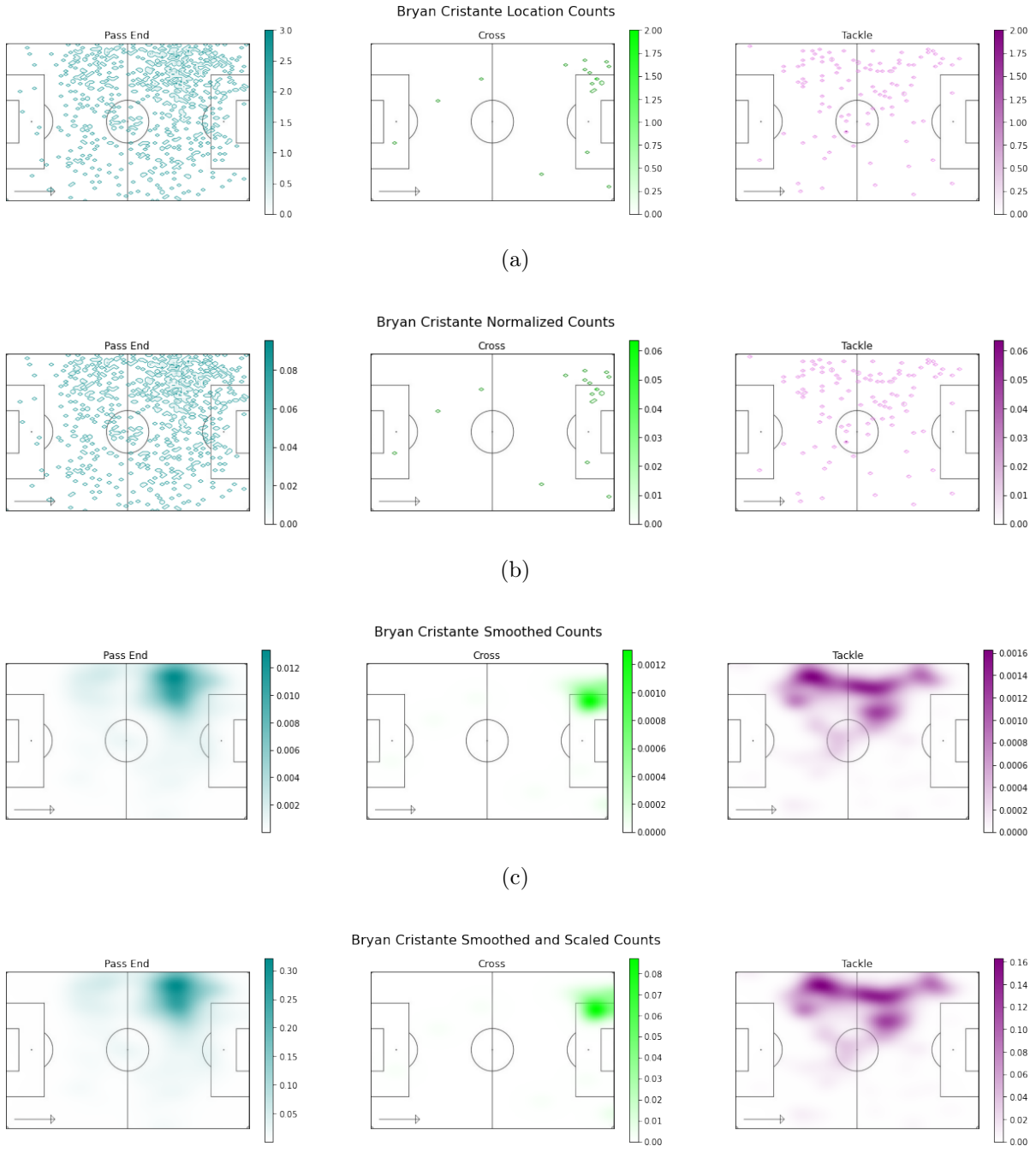


Figure 4.4: Bryan Cristante (Midfielder, AS Roma) Heatmaps Generation Process. In (a) the actions are collected and counted in a grid where each element represent a position in the field. In (b) the counts are normalized in function of the effective time played by the player along the season, in order to reproduce the action's frequency in 90 minutes (a generic match time). In (c) the frequencis are smoothed using a gaussian filter with smoothing parameter $\sigma = 5$ and kernel size implememtned as $K = 4\sigma$. In (d) the smoothed frequencies are scaled with a Min-Max Scaling, in order to better spread the data along the interval $[0, 1]$.

Chapter 5

Model Implementation

In this chapter the specific NMF model and VAE model used for the play-vectors extraction are explained, deeping into the architecture and the parameters used. In the first section the NMF-based approach is presented, while in the second one the VAE-based method is presented.

5.1 NMF Model Implementation

To apply Non-Negative Matrix Factorization to the data, i used `sklearn.decomposition.NMF`, a method from scikit-learn. One of the fundamental parameters of this model, as mentioned in Chapter 2, is the reconstruction error between the input matrix M and the reconstructed one $\tilde{M}$. Although the Frobenius Norm is usually used, especially in image processing, other error metrics can be used. The method used offers two other options:

- **Kullback-Leibler Divergence**: In this case, the original matrix is considered as a probabilistic distribution, and, as seen in the previous chapter, $D_{KL}[M||\tilde{M}]$ measures how inefficient it is to approximate M with $\tilde{M}$. Due to its probabilistic nature, it is often used when the data to be reconstructed represents frequencies.
- **Itakura-Saito Distance**: This measure focuses on reducing relative rather than absolute differences between M and M'. It is computed as

$$\mathcal{L}(M, \tilde{M}) = \sum_{i,j} \left(\frac{M_{ij}}{(\tilde{M})_{ij}} - \log \frac{M_{ij}}{(\tilde{M})_{ij}} - 1 \right),$$

  and is commonly used for audio signal processing.

Considering that the input data are heatmaps showing the frequencies of each game action, the Kullback-Leibler.
The aforementioned method takes other hyper-parameters as input. Below are the chosen values and their reasons for each of them.

- `init`: Determines the method for initializing the matrices W and H. By default, it is set to `init=random`, where the values of the matrices are set with a uniform distribution between 0 and 1; in my case, I chose `init=nndsvdar`, a method that uses the Non-Negative Double Singular Value Decomposition with Adjusted Random (NNDSVDAR [30]). This method uses SVD [31] to obtain an initial decomposition of W and H, and add a random noise to enhance the robustness of the model and improve convergence. This approach is particularly useful for sparse data.

- `solver`: This specifies the optimization algorithm. In my case, using Kullback-Leibler divergence as the reconstruction metric, it can only be set to `solver=mu`, which stands for Multiplicative Updating rule, a method discussed in the previous chapter.
- `alpha_W` / `alpha_H`: These parameters introduce an additional penalty E to the reconstruction, penalizing the algorithm if it uses excessively large values for W and H. They can take values between 0 and 1, and introduce regularization that can be formulated as

$$E_W = \alpha_W \cdot L_W(R)$$
$$E_H = \alpha_H \cdot L_H(R),$$

  where $L(R)$ quantifies the regularization, as described in the next point. In my case, I chose `alpha_W` $= 0.001$ and `alpha_H` $= 0.001$ to introduce slight regularization to the algorithm.
- `l1_ratio`: This parameter, formally denoted by R, regulates the trade-off between $L1$ and $L2$ regularization for the matrices as follows

$$L_W(R) = R\sum_{ij}|W_{ij}| + (1-R)\sum_{ij}(W_{ij})^2$$
$$L_H(R) = R\sum_{ij}|H_{ij}| + (1-R)\sum_{ij}(H_{ij})^2.$$

  In my case, I chose `l1_ratio` $= 0.5$ for a balanced trade-off between the two types of regularization.

In table 5.1 all the parameters chosen are summarized.

Table 5.1: Parameters for NMF Model

Parameter	Description	Value
`init`	Method for initializing matrices W and H.	`nndsvdar`
`solver`	Optimization algorithm used for factorization.	`mu`
`alpha_W`	Regularization parameter for matrix W.	0.001
`alpha_H`	Regularization parameter for matrix H.	0.001
`l1_ratio`	Trade-off between $L1$ and $L2$ regularization.	0.5

5.1.1 NMF Play-Vectors Extraction

This section explains the procedure for deriving the play-vectors for each player using NMF. There are two important preliminary considerations for this procedure. First, independent NMFs have been implemented for each selected feature. Therefore, each NMF has provided a compact version of a specific f-heatmap for each player $X(p, f)$. These compact representations are then concatenated to form the play-vector for each player. The second consideration concerns the features related to passes: indeed, the heatmaps of *PassStart* and *PassEnd* have been concatenated from the start, consistent with the work of Decroos and Davis [3], to extract a compact representation of passes in their entirety.
Now let's move to the detailed procedure. Assume we are working with a specific feature f. The dataset consists of $n = 1787$ heatmaps, with a shape of $(102, 102)$, as previously explained. Next, we linearize each heatmap $X(p, f)$ into a vector $x(p, f)$:

$$X(p,f) \longrightarrow x(p,f) \in \mathbf{R}_+^m,$$

where $m = 102 \cdot 102 = 10404$ is the number of pixels in a heatmap.
Once the heatmaps are linearized, they are concatenated into a matrix M^f:

$$\{x(p,f)\}_{p\in[1,n]} \longrightarrow M^f \in \mathbf{R}_+^{m\times n},$$

where M^f thus consists of n columns and m rows.
At this point, the iterative procedure begins to determine the ideal number of components k_f to decompose the matrix M^f into $W(k_f) \in \mathbf{R}_+^{m\times k_f}$ and $H(k_f) \in \mathbf{R}_+^{k_f\times n}$. Choose the optimal rank k_f is not a trivial task. In literature I found two different approaches. First one uses a simple statistical approach, quantifying the goodness in reconstruction and let space to human considerations to evaluate the best (or sufficient) rank. In the second one, an autonomous algorithm evaluate the best k. In this case many different procedures has been deveeloped recently. In [32] Guven et al use the Unit Invariant Knee Method (UIK) of the NMF on gene expression data sets, where the best rank is chosen looking for a sudden change in the average reconstruction error in function of k value. In [33] Cai et al determined the best rank k using a hypothesis testing approach combined with a likelihood ratio test. This second class of methods reach an high level of complexity, and are very powerful when the desired rank is way higher than 1. I decided to not increase the already present complexity of my thesis work and i used a simpler statistical approach: iterating over k_f, i computed the R^2 metric over the reconstructed $\tilde{M}^f$; once the level of reconstruction is considered sufficientely good, by an arbitrary threshold, the iteration is stopped at the optimal k_f. This method was evaluated enougth sufficient for the problem of my thesis work and, more over, setting an arbitrary threshold leave space for human considerations about the component chosen for each action type.
The R^2, also defined as determination coefficient, is computed as follows:

$$R^2(M^f, \tilde{M}^f) = 1 - \frac{RSS(M^f, \tilde{M}^f)}{TSS(M^f)}.$$

TSS is the Total Sum of Squares and is computed as

$$TSS(M^f) = \sum_{ij}(M_{ij}^f - \mu_{M^f})^2,$$

where μ_{M^f} is the mean value of input matrix M^f. RSS stands for Residual Sum of Squares and is computed as

$$RSS(M^f, \tilde{M}^f) = \sum_{ij}(M_{ij}^f - \tilde{M}_i^f j)^2,$$

and represent the sum of squared difference between the input and the reconstruction. The ratio between the error in reconstruction RSS over TSS, which is analogous to the variance of M^f, measure the badness of the output in function of the original sparsity. This ratio decrease with better outputs, and consequentially with higher ranks. R^2 subtract this ratio to 1. This quantity ranges from 0 to 1, and when it approaches 1, it indicates that a good portion of the variability in the original data is explained by the obtained approximation. Therefore, I set a threshold thr = 0.9, which ensure the goodness of the model and avoid the overfitting: if $R^2(k_f) < \text{thr}$, the procedure is restarted incrementing k_f by 1; if $R^2(k_f) \geq \text{thr}$, the obtained approximation sufficiently explains the variance of the original data, and we can exit the iteration.
Once the iteration is completed, we will have for the f-feature the corresponding matrices $T(k_f)$ and $Z(k_f)$, depending on the number k_f of components that meets the above criterion.

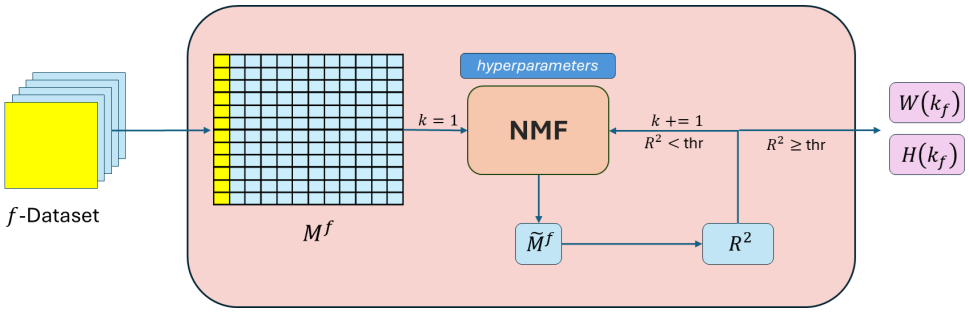


Figure 5.1: NMF implementation scheme. This figure represent the procedure implemented in order to evaluate the best number of components k_f for the factorization. Starting from $k_t = 1$ the algorithm produces W and H, where the last one is used to compute the Relative Explained Variance (R^2): if this measure is higher than a certain threshold we stop the the iteration; if it is below the threshold we start over increasing k_f of 1, until we get a sufficent R^2.

The compact representations of players for the f-feature are contained in $H(k_f)$. In fact each column of $H(k_f)$ is the compact representation of the relative M^f column. We will refer to the $H(k_f)$ p-th column as $z_{NMF}(p, f)$, which is basically the *sub*-play-vector for the player p and feature f obtained by NMF. Applying this procedure to all f-datasets, and concatenating the relative *sub*-play-vector for each player, we get the wanted complete NMF *play-vectors* $z_{NMF}(p) \in \mathbb{R}_+^{L_{NMF}}$, where L_{NMF} is the sum of the number of k_f components, and the actual dimensionality of the play-vectors:

$$L_{NMF} = \sum_t k_f.$$

We will refer to the set of NMF-play-vectors $z_{NMF}(p)$ with Z_{NMF}. The play-vectors assembling procedure described above is summarized in figure 5.2.

5.2 VAE Model Implementation

In this section, I describe the architecture of the VAE used to project heatmaps into a latent space, from which I derived new PlayerVectors. I will explain the extraction process and the different layers of the structure in detail.
It's important to make an introductory consideration. Autoencoders can take multi-channel input data. For example, traditional autoencoders used for image classification ([34], [35], [36]) take three input channels corresponding to red, green, and blue (RGB). Initially, I implemented six independent VAEs, one for each type of heatmap. However, this approach not only limited the potential of the VAE but also required enormous training time. For this reason, I decided to implement a VAE that takes six-channel input: instead of a single type of heatmap, the input consists of all six heatmaps concatenated together. Specifically, to implement the algorithm's training, I used `pytorch` [37], with batches of 32 data points. For simplicity, from here on, the use of batches is implied, and I will refer to the input as a single block of six concatenated heatmaps.
Once the dataset is assembled with concatenated heatmaps, the data passes through the encoder, which is made up of three blocks, structured as follows: a convolutional layer (`torch.nn.Conv2D()`), which applies a 2D convolution, using a kernel to extract spatial features; a Dropout layer (`torch.nn.Dropout(0.1)`), which randomly sets 10% of the neurons to 0 during training to prevent overfitting; finally, `torch.nn.ReLU()` sets the Rectified Linear Unit as the activation function, described as

$$f(x) = \max(0, x),$$

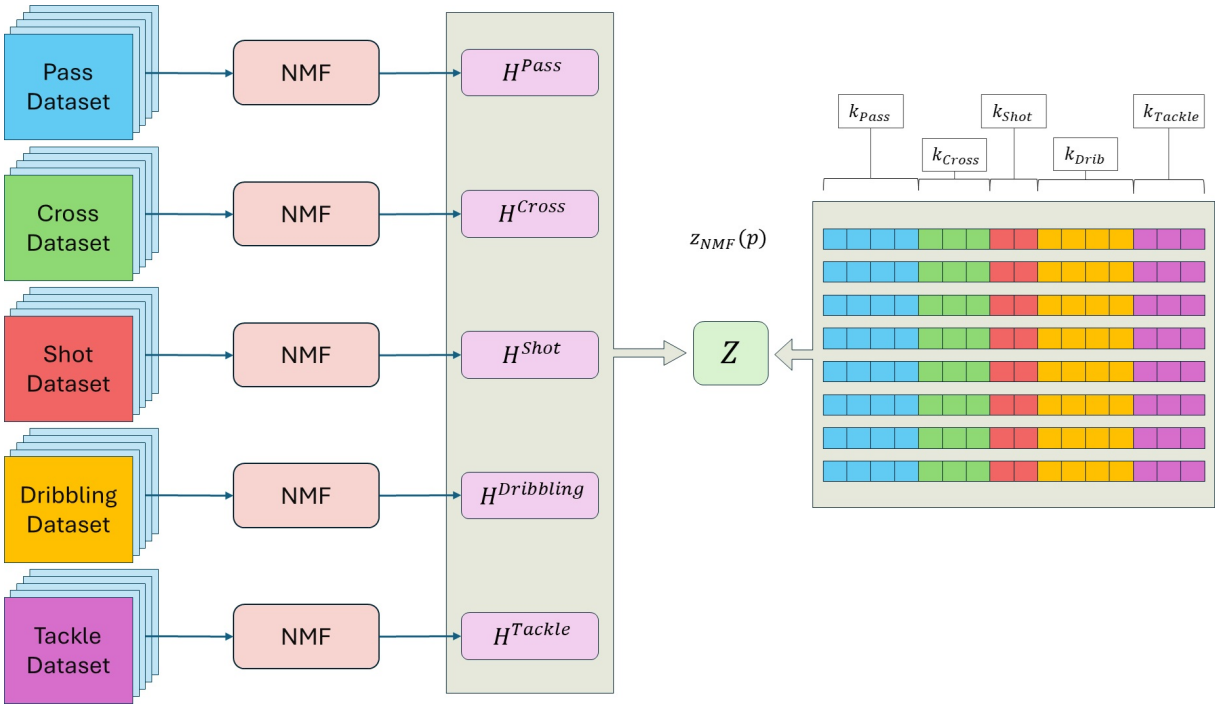


Figure 5.2: General scheme for the assembling of play-vectors dataset by NMF algorithm. For each feature is retrieved the H^f matrix which contains the weights of the f-components. Concatenating the weights the NMF play-vectors are obtained.

which accelerates convergence and introduces sparsity into the data by zeroing out negative values, making the model more robust and efficient.
After passing through the three convolutional blocks, we obtain data composed of 32 channels with a shape of $(13, 13)$. This is flattened, using the `torch.nn.Flatten()` function, into a vector of $32 \cdot 13 \cdot 13 = 5048$ elements. This is the first layer of a neural network (NN) consisting of a second layer of 64 nodes, a third layer of L_{VAE} nodes, and finally, two parallel layers, each composed of L_{VAE} nodes: these are responsible for computing $\vec{\mu}$ and $\log \vec{\sigma}^2$, as explained in the previous chapter.
The general architecture of the encoder is schematically shown in figure 5.3, while a more detailed explanation of the layers is presented in table 5.2.

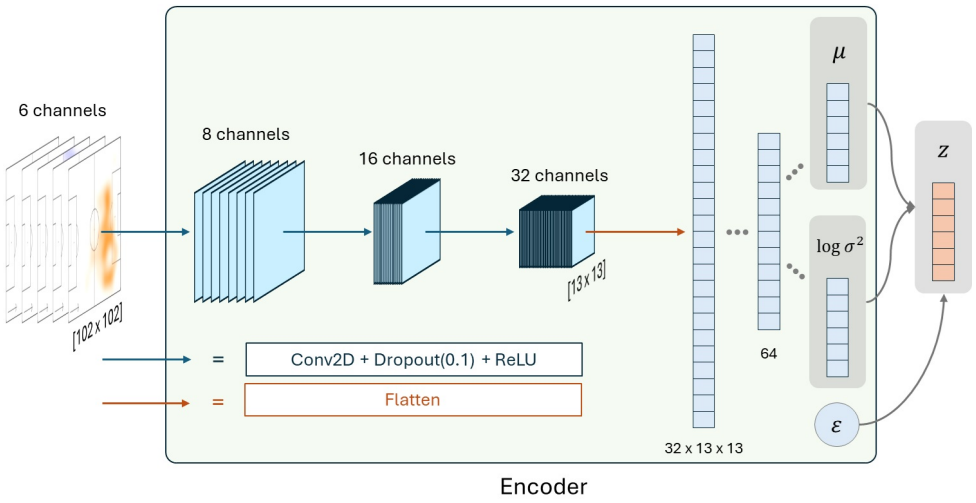


Figure 5.3: VAE Encoder Architecture used in my thesis work.

	Type	Parameters	Output Shape
1°C	`torch.nn.Conv2d`	`in`= 6, `out`= 8, `kernel`= [3, 3], `stride`= 2, `padding`= 1	[8, 52, 52]
	`torch.nn.Dropout`	0.1	
	`torch.nn.ReLU`	-	
2°C	`torch.nn.Conv2d`	`in`= 6, `out`= 8, `kernel`= [3, 3], `stride`= 2, `padding`= 1	[16, 26, 26]
	`torch.nn.Dropout`	0.1	
	`torch.nn.ReLU`	-	
3°C	`torch.nn.Conv2d`	`in`= 6, `out`= 8, `kernel`= [3, 3], `stride`= 2, `padding`= 1	[32, 13, 13]
	`torch.nn.Dropout`	0.1	
	`torch.nn.ReLU`	-	
	`torch.nn.Flatten`	-	5408
1°L	`torch.nn.Linear`	`in`= 5048, `out`= 64	64
	`torch.nn.ReLU`	-	
2°L	`torch.nn.Linear`	`in`= 64, `out`= L_{VAE}	L_{VAE}
	`torch.nn.ReLU`	-	
μ	`torch.nn.Linear`	`in`= L_{VAE}, `out`= L_{VAE}	L_{VAE}
$\log \sigma^2$	`torch.nn.Linear`	`in`= L_{VAE}, `out`= L_{VAE}	L_{VAE}

Table 5.2: Parameters used in the VAE Encoder layers. 1C, 2C, 3C represent the convolutional blocks; 1L, 2L represent the linear blocks; in orange are highlighted the linear blocks which leads to the computation of $\vec{\mu}$ and $\log \vec{\sigma}^2$.

As explained in the subsection "Reparametrization Trick," the vectors $\vec{\mu}$ and $\log \vec{\sigma}^2$ are used to generate $\vec{z}$, the projection of the data onto the reduced dimensionality latent space L_{VAE}.
After obtaining the projections onto the latent space, the data enter the decoder, which brings them back to the original dimensionality. First, they go through a neural network consisting of two linear layers with 64 and 5048 nodes, respectively, and are then reshaped using `torch.nn.Unflatten()` to a shape of $(32, 13, 13)$. From here, they pass through three convolutional blocks, each consisting of two layers. The first layer is a deconvolutional layer (`torch.nn.ConvTranspose2D`()), which increases the spatial dimensions while reducing the number of channels. Following this, an activation function is applied: the first two blocks use `torch.nn.ReLU`, as seen previously in the encoder; the final deconvolution uses `torch.nn.Sigmoid()`, which applies the sigmoid activation function

$$f(x) = \frac{1}{1 + e^{-x}}.$$

This function is commonly used at the end of the decoder for its intrinsic probabilistic representation, returning an output between 0 and 1. This feature makes it ideal for reconstructing heatmaps, which contain frequency values.
The decoder architecture is schematically represented in figure 5.4, while a more specific explenation of the layers is presented in table 5.3.

5.2.1 VAE Training and Play-Vectors Extraction

In this case, no separate procedure was needed to extract the play-vectors as in the case of NMF, since all types of heatmaps are processed simultaneously and projected into the latent space during training. Therefore, once the model is trained, it is simply necessary to filter the original data using the trained encoder. Different values for L_{VAE} was tried to achieve the best

	Type	Parameters	Output Shape
1°L	`torch.nn.Linear`	`in`= L_{VAE}, `out`= 64	64
	`torch.nn.ReLU`	-	
2°L	`torch.nn.Linear`	`in`= 64, `out`= 5408	5408
	`torch.nn.ReLU`	-	
	`torch.nn.Unflatten`	`dim`= 1, `unflattened_size`= (32, 13, 13)	[32, 13, 13]
1°CT	`torch.nn.ConvTranspose2d`	`in`= 32, `out`= 16, `kernel`= [3, 3], `stride`= 2, `padding`= 1, `output_padding`= 1	[16, 26, 26]
	`torch.nn.ReLU`	-	
2°CT	`torch.nn.ConvTranspose2d`	`in`= 16, `out`= 8, `kernel`= [3, 3], `stride`= 2, `padding`= 1, `output_padding`= 0	[8, 52, 52]
	`torch.nn.ReLU`	-	
3°CT	`torch.nn.ConvTranspose2d`	`in`= 8, `out`= 6, `kernel`= [3, 3], `stride`= 2, `padding`= 1, `output_padding`= 1	[6, 102, 102]
	`torch.nn.Sigmoid`	-	

Table 5.3: Parameters used in the VAE Decoder layers.

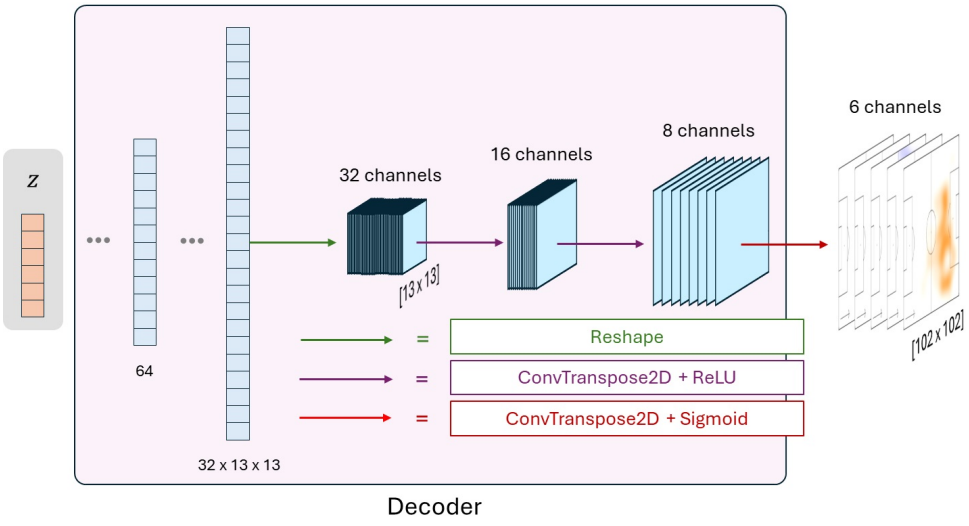


Figure 5.4: VAE Decoder Architecture used in my thesis work.

Optimizer	
Method	`torch.optim.Adam()`
learning rate	10^{-3}
λ	$5 \cdot 10^{-5}$
β_1, β_2	0.9, 0.999
Training	
epochs	500
batch size	32

Table 5.4: Optimizer and Training Parameters used in the VAE model.

reconstruction in output. A value of $L_{VAE} = 20$ was choosen. Not only to compare properly the power of both models with the same latent sapce dimensionality: in fact, higher values has been tried, up to 150, but increasing the dimensionality leaded to unused dimensions, and, overall, it did not significantly improve reconstruction quality.
The loss function used is the one described in section 3.2 from equation 3.4. The first term was computationally translated using the Binary Cross Entropy (BCE), defined as

$$BCE = -\frac{1}{N}\sum_{ij} X_{ij}\log\Big(\tilde{X}_{ij}\Big) + (1 - X_{ij})\log\Big(1 - \tilde{X}_{ij}\Big), \tag{5.1}$$

where X_{ij} and $\tilde{X}_{ij}$ represent the elements of the input and reconstructed data, respectively, and N is the total number of elements in each data point. This definition derives from the following probabilistic interpretation. In the context of a VAE, we can consider $\tilde{X}_{ij}$ as the probability (predicted by the model) that the corresponding pixel of the original data is active ($X_{ij} = 1$). Assuming this probability is described by a Bernoulli distribution, we get

$$p(X_{ij}) = {\tilde{X}_{ij}}^{X_{ij}} \Big(1 - \tilde{X}_{ij}\Big)^{1-X_{ij}}.$$

Similarly to the term analyzed in equation 3.2, we apply the logarithm

$$\log p(X_{ij}) = X_{ij}\log\tilde{X}_{ij} + (1 - X_{ij})\log(1 - \tilde{X}_{ij}).$$

Expressing the expectation as the mean over the elements, we obtain the definition 5.1.
The second term related to the Kullback-Leibler divergence is computationally translated, as seen in section 3.2, from 3.10.
The optimizer used is Adam, which stands for Adaptive Moment Estimation, and it is widely used in Deep Learning model training. It combines the capabilities of two well-known optimizers: AdaGrad, from which it takes an adaptive learning rate for each parameter, using a smaller learning rate for parameters updated more frequently, and RMSProp, from which it takes an optimal balance in the learning rate using an exponential moving average, both for the gradient and its square. Let's see how Adam works in detail. First, it computes the gradient of the loss function

$$g_t = \nabla_{\theta_t}\mathcal{L}_{VAE}(\theta_t),$$

where θ_t indicates the parameters of the VAE, updated at iteration t. Then it updates the moment m_t and the variance v_t as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2,$$

where β_1 and β_2 are constants, typically set to 0.9 and 0.999, respectively. In the first iterations, the estimates of m_t and v_t tend to be biased toward 0, so the following correction is applied:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

where the denominator terms tend to 1 as the iterations progress. Finally, the parameters θ are updated with the following rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \varepsilon}}\hat{m}_t,$$

where α is the learning rate, and ε, a small positive value, is added to the denominator to avoid division by zero. In my specific case, I added a weight decay regularization term which is added

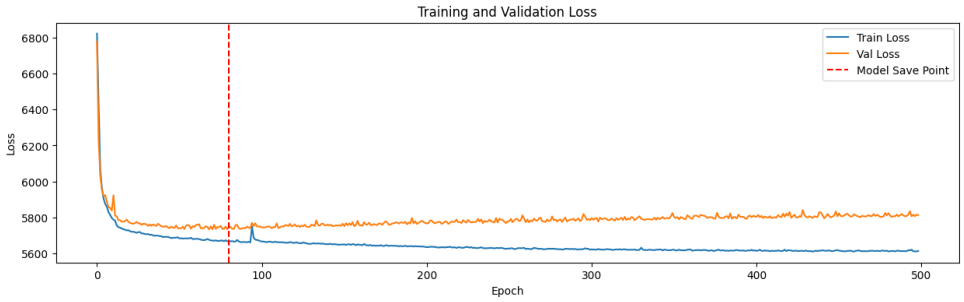


Figure 5.5: VAE Training. Blue and orange lines represent respectively the training and validation loss. The red dashed red line represent the epochs relative to the best model. From the plot can be seen that, while the training loss get lower, the validation loss get higher: this stage is called overfitting, where the model lose its generalist ability to reconstruct external data.

at each iteration to the loss function as follows:

$$\mathcal{L}_{tot} = \mathcal{L}_{VAE} + \lambda||\theta||^2,$$

where λ is the weight decay factor, which control the intensity of regularization.
The specific parameters used in the VAE training are exposed in table 5.4. In training stage, i always used a validation set to evaluate the generalist capacity of the model and avoid overfitting. Along the procedure, at each iteration, i saved the model parameter every time the loss on the validation set get a lower result. The figure 5.5 visualize the VAE training: the blue and orange line represent respectively the training and validation loss; the dashed red line represent the epochs relative to the best model. From the plot can be seen that, while the training loss get lower and lower (indicating a best accuracy in training data reconstructions) the validation loss get higher: as mentioned before, this stage is called overfitting, where the model lose the generalist ability to reconstruct external data.
As anticipated previously, once the VAE is trained has been possible to extract play-vectors for all 1787 players just feeding the trained encoder with the heatmaps, as shown schematically in figure 5.6.

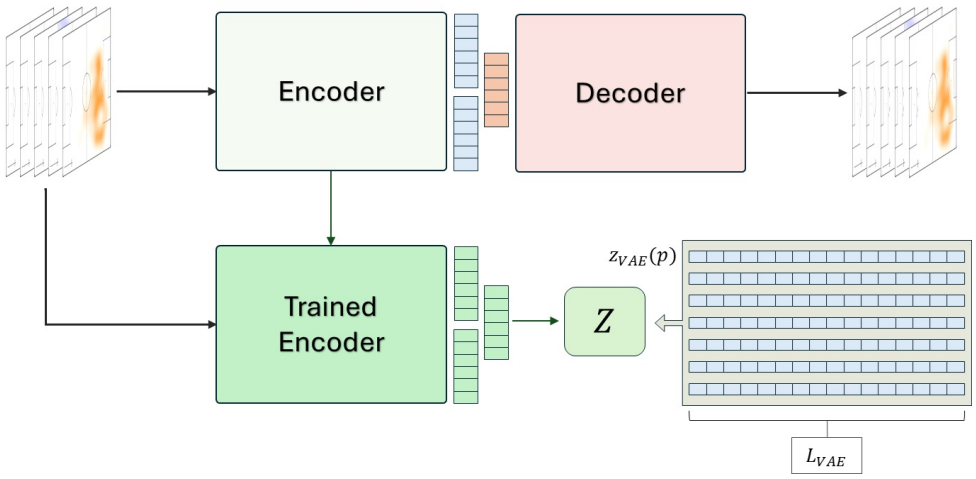


Figure 5.6: Schematic representation of play-vectors extraction with VAE training. Once the VAE training is complete, all the 1787 block of heatmaps (composed by the concatenated 6-features heatmaps) are processed by the trained encoder and projected in the latent space Z. Then simply Z is the set of the wanted 1787 play-vectors $z_{VAE}(p) \in \mathbb{R}^{L_{VAE}}$.

Chapter 6

Analysis

In this chapter are exposed the analysis on the vectors obtained from both tecniques. Firsly the representativeness of play-vectors are quantified by the *Player Retrieval Analysis*. Once the best play-vectors in accuracy are chosen, these will be use to investigate on specific pattern of playing style detection using the DBSCAN clustering.

6.1 Player Retrieval Analysis

The first analysis I carried out, inspired by the work of Decroos et al. [3] and Hyeonah et al. [12], was the Player Retrieval Analysis (PRA). This allows, given certain assumptions, to assess the effectiveness (or more precisely, the representativeness) of the play-vectors obtained from the developed models. The basic idea is to derive, for each player p, two play-vectors that, despite some slight differences, reflect the playing style of the same player. To achieve this, Hyeonah et al. split the original event dataset into the first half of the season and the second half, processed the related heatmaps, and then extracted the play-vectors. I decided to proceed differently. Players during the season have periods of good form and others of decline, not to mention those who get injured. Often, players find themselves playing under different coaches who place them in different roles, or many players experience turnover due to cup competitions. All these reasons suggest that comparing playing style between the first and second half of the season might not be optimal, as there is a good chance that a player does not maintain the same playing style. Instead, for this analysis, we need two vectors for the same player that are representative of the same playing style, or at least as similar as possible. Therefore, I decided to divide my original dataset into two sub-datasets, one related to events that occurred in the first halves and the other related to events in the second halves of the matches. In this way, for each player p and each feature f, two heatmaps were obtained: $X^{1H}(p, f)$ representing the frequencies of the first halves and $X^{2H}(p, f)$ representing the frequencies of the second halves.
Both the NMF and VAE models described in Ch.5 were trained using data from the first half to obtain the play-vectors for the first half $z^{1H}(p, f)$. The same trained models were then used to process the second-half data and extract the second-half play-vectors $z^{2H}(p, f)$. Extracting $z^{2H}(p, f)$ by training the models directly on second-half data was avoided, as this could not only lead to different dimensions for $z^{1H}(p, f)$ (for example, in the case of NMF, where the final dimension depends on the optimization process), but also to maintain consistency in the input data transformations leading to the dimensionality reduction of the heatmaps. Figures 6.1 and 6.2 outline the procedures for obtaining the play-vectors through NMF and VAE, respectively.

In the case of NMF, after reconstructing the input matrices for each feature $M^{f,1H}$, $H^{1H}(k_f)$ was derived, which contains the first-half play-vectors $z^{1H}_{NMF}(p, f)$ and thus corresponds to

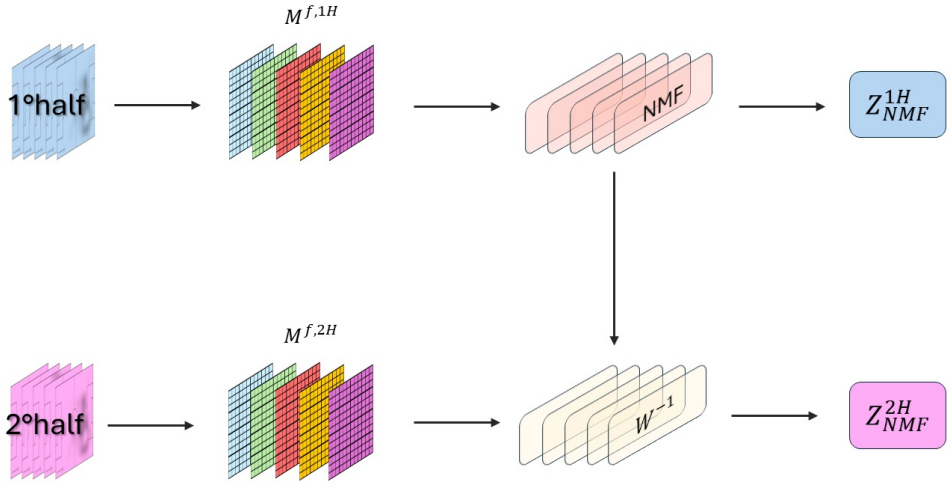


Figure 6.1: NMF-based 2^{nd} half play-vectors extraction. 1^{st} half heatmaps are used to obtain, by five different NMFs, the relative play-vectors. Then the component matrices $W(k_f)$ are inverted and multiplied to $M^{f,2H}$ to compute the 2^{nd} half play-vectors.

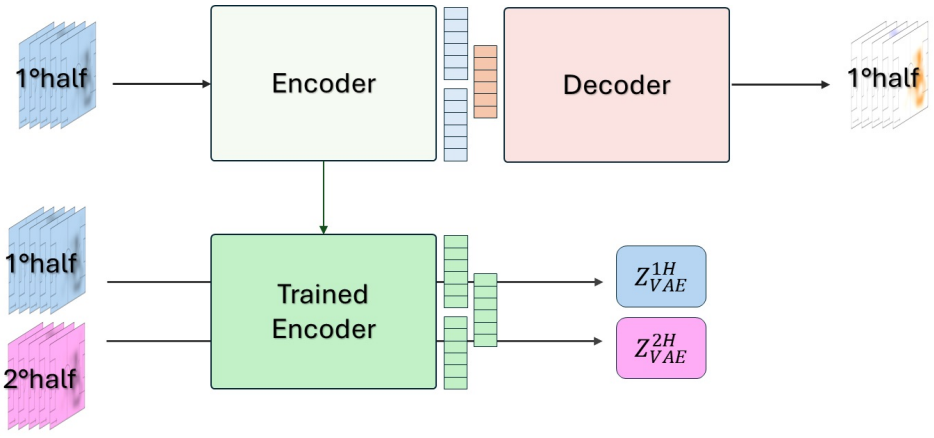


Figure 6.2: VAE-based 2^{nd} half play-vectors extraction. Once the VAE is trained with 1^{st} half heatmaps, the 2^{nd} half ones are mapped with it to the latent space in order to get the 2^{nd} half play-vectors.

Action Type	Pass	Cross	Shot	Dribbling	Tackle
NMF Components	5	3	3	4	5

Table 6.1: Number of NMF components obtained for each action type.

Z^{1H}_{NMF}, along with the matrices containing the transformations according to the principal components $W^{1H}(k_f)$. The latter were inverted and used, through dot product with $M^{f,2H}$, to derive $H^{2H}(k_f)$, which contains the second-half play-vectors $z^{2H}_{NMF}(p, f)$ and thus corresponds to Z^{2H}_{NMF}. Applying the Relative Explained Variance procedure, exposed in the previous chapter, i got the k_f values presented in table 6.1. In the case of the VAE, once the model was trained with first-half data, the trained encoder was extracted to project the heatmaps of both the first and second halves into the latent space, obtaining the corresponding Z^{1H}_{VAE} and Z^{2H}_{VAE}. Since the analysis is similar for both approaches, NMF and VAE, we now generalize by referring to the play-vector as $z^t(p)$ and to the related sets as Z^t, where $t \in [1H, 2H]$. Given a player p, a distance function is applied between $z^{1H}(p)$ and all the play-vectors in Z^{2H}, obtaining a distance vector, where the minimum values indicate the play-vectors $z^{2H}(q)$ most similar to $z^{1H}(p)$, where q indexes the players in Z^{2H}. The metrics used in my thesis are MSE, Manhattan distance (MD) and Cosine distance (CD). Manhattan distance is defined as

$$d_{MD}(v, u) = \sum_{i=1}^{L} |v_i - u_i|,$$

where u and v are two generic vectors of lenght L, while Cosine distance is defined as

$$d_{CD}(v, u) = 1 - \frac{v \cdot u}{||v||||u||},$$

where the same notation as Manhattan distance has been used. The latter is derived from Cosine Similarity but revisited so that the degree of similarity does not have negative values. In this way, for each index p, a corresponding distance vector $D(q)$ is obtained:

$$D(p) = [d(p, q = 1), d(p, q = 2), ..., d(p, q = n)].$$

Once $D(p)$ is sorted in ascending order by distance, obtaining $D^{ord}(p)$, we extract the list Q of the corresponding indices: this then contains the indices from 1 to n, ordered by the similarity between $z^{1H}(p)$ and $z^{2H}(q)$. Fixing an arbitrary parameter K, we check if $q = p$ in the first K indices of Q. If this condition is met, we increment a counter by one.
This procedure is performed for all indices p, and the total is divided by n: in this way, the PRA returns an accuracy a which measures how much each player shows similarity (parameterized by K) between their two play-vectors $z^{1H}(p)$ and $z^{2H}(p)$. It is assumed that higher accuracy indicates that the play-vector, and the related model, is a good representation of the player's playing style. The accuracy can be mathematically formulated as follows:

$$a = \frac{1}{n} \sum_{p=1} \mathbb{I}[p \in Q_K(p)],$$

where $Q_K(p)$ indicates, given p, the indexes list of its K-most similar play-vectors in Z^{2H}. The analysis just described is schematically shown in figure 6.3.

In my analysis I obtained the accuracies for both approaches, NMF-based and VAE-based, using $K = 1, 3, 5, 10, 15, 20$ and the three metrics mentioned before. In figure 6.4, it is possible to see the comparison between the accuracies, where each metric is represented by a different color, while the dashed line indicates the model used to obtain the play-vectors: dashed for the VAE

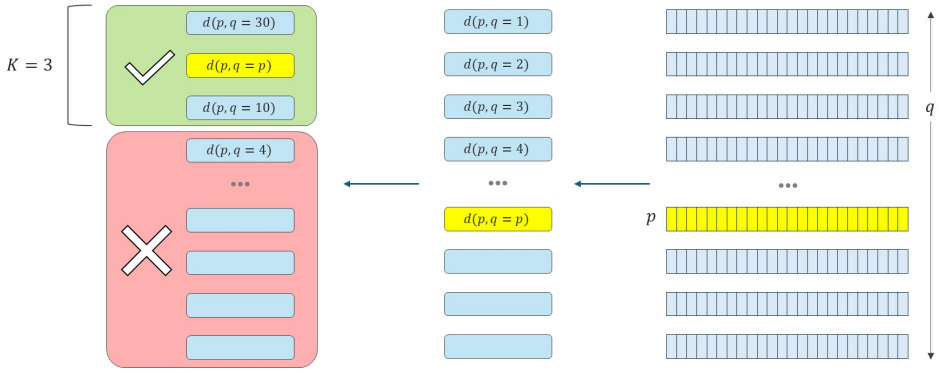


Figure 6.3: Player Retrieval Analysis. A 1^{st} half time play-vector of a player p is selected and the distance with all the vectors, indicized by q, in 2^{nd} set are computed. The distances are sorted in ascending order. Given a parameter K we check for the first-K q-indexes, collected in the list $Q_K(p)$: if p is in $Q_K(p)$ then we count it as success. The procedure is computed for each player, increasing a counter of 1 for each success. Then the mean accuracy is calculated dividing the counter value by the number of players.

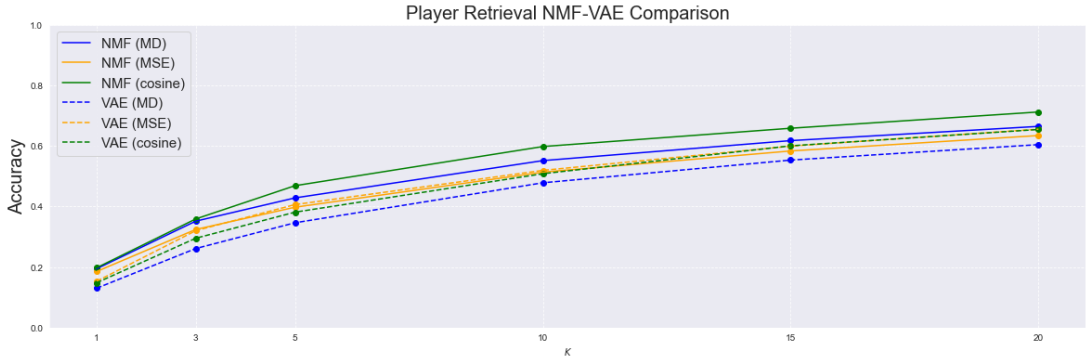


Figure 6.4: NMF (continuos line) and VAE (dashed line) Player retrival Accuracies are compared for three different metrics, Manhattan distance (blue lines), Mean Squared Error (orange lines) and Cosine distance (green lines).

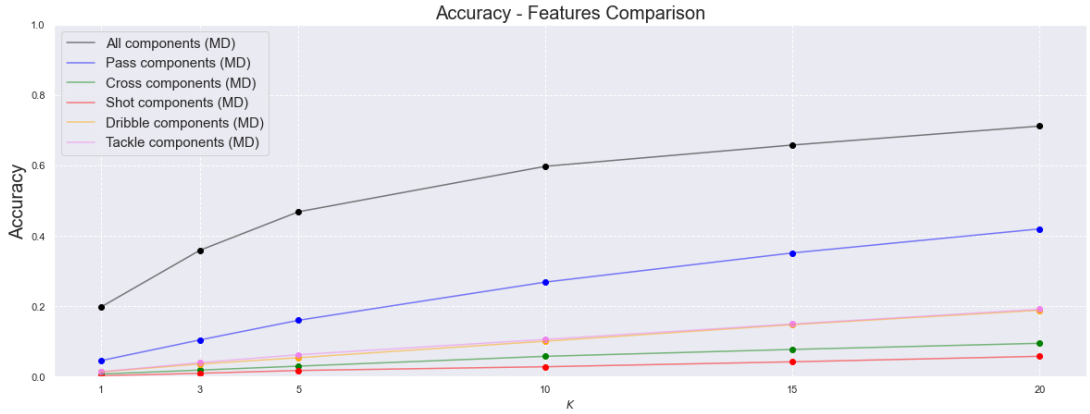


Figure 6.5: Accuracy for each feature sub-play-vector set $Z^t_{NMF}(f)$.

and solid for the NMF. It is clear how the vectors obtained from the NMF get a higher accuracy for all values of K. In particular, how highlighted by the table 6.2, when we compute the accuracies using the cosine distance (green solid line in the graph in figure). This gives us some points for observations. Despite the play-vectors obtained from the VAE having almost five times the variables available compared to those obtained with the NMF, they show less effectiveness in characterizing the players. Probably the difficulties encountered by this approach is in the probabilistic nature of this model: it is known that the reconstructions of autoencoders, especially variational ones, tend to return images that are particularly blurry. This happens due

Method, Metric	$K = 1$	$K = 3$	$K = 5$	$K = 10$	$K = 15$	$K = 20$
NMF, Manhattan distance	0.194	0.351	0.428	0.551	0.617	0.664
NMF, Mean Squared Error	0.185	0.324	0.397	0.513	0.583	0.634
NMF, Cosine distance	0.198	0.359	0.468	0.598	0.658	0.712
VAE, Manhattan distance	0.130	0.261	0.346	0.478	0.553	0.604
VAE, Mean Squared Error	0.153	0.320	0.406	0.519	0.599	0.655
VAE, Cosine distance	0.148	0.295	0.381	0.508	0.600	0.654

Table 6.2: Comparison of accuracies between NMF and VAE, using three different metrics: Manhattan distance, Mean Squared Error and Cosine distance. The green row highlights the model and the metric which gave the best accuracies in function of K.

to the probabilistic sampling in the reconstruction phase. A direct consequence of this, related to my analysis, is that players with similar playing styles, despite the differences, end up being approximated in the same reconstruction, thus giving generality to their projections in the latent space. This generality translates into lower accuracy in the retrieval analysis.
Secondly, since the best model is the one based on NMF, I wanted to evaluate in detail the contribution of each single feature. To do this, I applied the same procedure to each individual feature. For each one, I took the sets related to the first and second halves with the sub-play-vectors $Z^t_{NMF}(f)$ of length k_f and obtained the accuracy through Cosine distance. The results obtained are shown in figure 6.5. It can be seen how the passes guarantee the highest accuracy when used alone, followed by the Tackle and Dribbling features; the least contribution is given by Cross and Shot. We can notice that the most effective action types are all actions that usually cover the whole field, while Cross and Shot are usually projected to attack and in fact mainly occur around the penalty area. For these reasons I wanted to measure the sparsity of each feature along the heatmaps and quantify it to see if there is correlation with the accuracy degree of the sub-play-vectors. To do this, I collected all the heatmaps for action type and summed the frequencies spatially, obtaining a single heatmap $X(f)$ for each feature. In information theory,

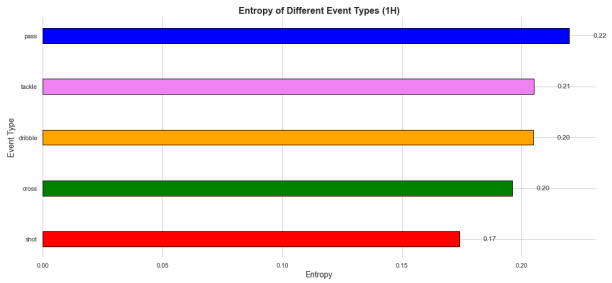


Figure 6.6: Shannon entropy computed for each feature.

the sparsity of frequencies in a matrix is measured by the Shannon Entropy E, defined as

$$E_f = -\sum_{ij} X_{ij}(f) \log X_{ij}(f), \tag{6.1}$$

where i and j are the indices on which the elements of $X(f)$ run and E_f represents the entropy computed for that matrix. The values of the entropies are shown in figure 6.6 and it can clearly be seen the correlation with the accuracies in figure 6.5: the entropies shown in the graph are arranged vertically in descending order, and it is the same vertical order that the accuracies follow for each K. This result confirms not only, as said before, that the spatial distribution infers in the effectiveness of a feature in characterizing a player through play-vector, but that it was a good choice to include Tackles as a feature, despite in the work of Decroos et al it being considered a defensive event, a playing style that depends mostly on the position in the field. As explained in the second chapter, I agree only in part with this view: if we wanted to describe only the defensive playing style, tackles would not be sufficient to describe it; however, since my analysis is more general, it makes sense to consider as much information as possible, assuming that it makes sense. Lastly, i wanted to comper my PRA results with the ones obteined by Decroos et al and Hyeonah et al. A comparison of the accuracies is visualized in figure 6.7 and reported in the details in table 6.3. To make this comparison, I first needed to adapt my dataset to theirs. In previous papers, they worked with a much smaller number of players, selecting only those who played at least 900 minutes during the whole season. With this filtering, they obtained a dataset of 809 players. It is obvious that the more players there are, the harder it is to identify them in the entire dataset. So, I proceeded as follows. I ran 100 simulations. For each simulation, I randomly selected 809 players and used their play-vectors to implement the PRA using the play-vectors obtained with NMF and using the Cosine distance as a metric (which gave the best result in the previous analysis). After running the 100 simulations, I calculated an average accuracy for each K. As can be seen, with my approach, as K increases, the percentage gap with the accuracies obtained by Decroos et al. remains stable, while the gap with those obtained by Hyeonah et al. increases. This trend, better shown in Figure 6.8, confirms the validity of this model and the work done.

Method, Metric	**1**	**2**	**3**	**4**	**5**	**6**
my NMF, Cosine distance	0.283	0.501	0.602	0.726	0.798	0.849
Hyeonah et al, Manhattan distance	0.294	0.456	0.543	0.666	0.724	0.765
Decroos et al, Manhattan distance	0.225	0.410	0.509	0.637	0.711	0.753

Table 6.3: Comparison of my NMF accuracies and Decroos et al [3] and Hyeonah et al [12] ones.

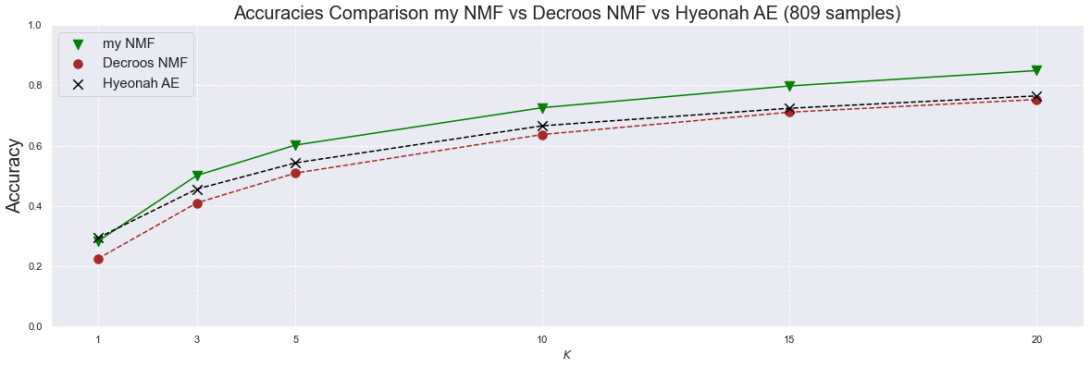


Figure 6.7: Player Retrieval accuracies obtained by my NMF approach compared with Decroos et al [3] and Hyeonah et al [12] results.

6.2 Clustering Analysis

In this section, the clustering analysis performed on the play-vectors obtained from the Non-Negative Matrix Factorization is presented, as they have proven to best characterize the playing style of the players. The chosen clustering algorithm is DBSCAN, which stands for Density Based Spatial Clustering Algorithm with Noise, and it groups samples in space based on the density of nearby samples in an iterative way. The goal of the DBSCAN algorithm is to group the reduced-dimension representations obtained via NMF and identify playing patterns among players. The clustering pipeline applied to the vectors includes a sample processing phase, followed by the application of DBSCAN Clustering. To provide a clear understanding of the procedure, this section first gives a brief introduction to the DBSCAN algorithm, followed by an explanation of the EDDA procedure, which stands for Enhanced DBSCAN with Density Attractors. Finally, the results of this analysis and the sub-patterns found between playing styles are presented.

6.2.1 Overview of the DBSCAN Algorithm

DBSCAN, introduced by Ester in 1996 [38], is a density-based clustering algorithm. Unlike other algorithms, DBSCAN does not require the final number of clusters as input, but instead, it can determine the number of clusters by itself through an iterative process. One of its strengths is its ability to recognize clusters of different shapes, as its grouping criterion is based on spatial density. To do so, DBSCAN requires two parameters: ε (epsilon), which represents the minimum distance for two samples to be considered *neighbors*, and $minPts$, which is the minimum number of samples required to form a cluster. Using these two parameters, the algorithm classifies samples into three different categories:

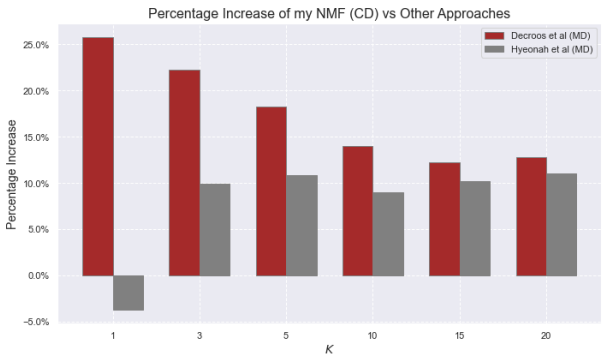


Figure 6.8: Percentage Increase of my NMF accuracies relative to Decroos et al [3] and Hyeonah et al [12] results.

- **core point**: a sample that has at least $minPts$ in its ε-neighborhood.
- **border point**: a sample that is near a core point but does not have enough neighbors to be considered a core point.
- **noise point**: a sample that is neither close to any core points nor has enough neighbors to be considered a core point.

The procedure for defining clusters occurs iteratively. A random sample from the dataset is selected. It is checked against the parameters to see if it is a core point: since there are no other points at the start of the procedure, this sample can either be a core point or a noise point. If it is a noise point, it is labeled as such, and the algorithm moves on to select another point randomly. If it is a core point, all the samples in the neighborhood are labeled as belonging to the same cluster. For each of these points, the same process is applied, expanding the cluster. The expansion stops if the algorithm reaches a border point: in this case, starting from that sample, no further expansion occurs. By doing this, DBSCAN is able to deterministically create a set of clusters (and noise, labeled separately) based on the two parameters ε and $minPts$.
It is important to understand the effects of parameter variation on the clustering results. ε, by regulating the neighborhood size of each sample, acts as a connector between points: the larger its value, the higher the probability that spatially separated samples will be connected and belong to the same cluster. $minPts$ controls the minimum number of samples needed to define a cluster and acts as a spatial discriminant. Increasing its value imposes a stricter criterion for forming a cluster, requiring higher spatial density. As a result, more coherent and dense clusters are obtained, but the number of outliers (noise) increases. Decreasing $minPts$ relaxes this constraint, leading to more expansive clusters with less noise but a higher risk of obtaining less coherent and more generalized clusters.
There is no definitive procedure for selecting these two parameters beforehand. Usually, they are chosen through trial and error, along with intuition. However, an interesting technique involves a prior analysis of the distances between samples: the Elbow (or Knee) Method, presented for the first time by Thorndike in [39]. This technique uses a parameter K_{nn}, representing the K-th *nearest neighbor* for a given sample. For each sample p, the distance $d_{K_{nn}}(p)$ is computed, and the list of these distances is sorted in descending order: this way, the samples characterized by high spatial density are listed at the top, and those with lower density are at the bottom. The goal is to find the point where this distance increases sharply: this point, taken as ε^* (optimal epsilon), represents the distance that separates high-density samples from low-density ones. For consistency with the DBSCAN parameters, K_{nn} is usually set equal to $minPts$. It is called the elbow method because the distance graph presents an "elbow": a point where the relative distances sharply increase. Identifying the sample corresponding to the elbow is not always simple. In my thesis work, I used the Python package `knee.KneeLocator` [40], but it almost always returned an ε^* slightly higher than the true value, which is visible by eye from the distance graph. For this reason, I consistently decreased the epsilon value by a reduction factor f_ε in the range $(0.4, 0.6)$.

6.2.2 EDDA Pipeline

The pipeline described in this section is my personal innovative approach, implemented to help DBSCAN detect clusters of samples that are characterized by a lot of noise in the original dataset. Specifically, it helps the clustering algorithm detect distinct sub-clusters, characterized by intermediate samples that would influence classic DBSCAN, which might enclose both sub-clusters into a single large cluster. This is a specific approach designed to detect sub-patterns in the heatmaps analyzed in my thesis work: one of the most interesting aspects of analyzing players' playing styles is observing distinct characteristics among players labeled under the same general role (such as two defenders, where one plays as a right center-back, while the other is

a left-back) or similarities in playing style between two roles traditionally considered different (like an offensive right-back and a wide midfielder, both involved in the offensive and defensive phases of the right side). As we will see in the results subsection, this pipeline overcomes the limitations of classic DBSCAN in detecting sub-groups in the original dataset and, at the same time, significantly reduces noise. This last aspect is not necessarily a strength, as outlier analysis is fundamental in clustering to detect samples with atypical characteristics, and it is not always positive to label them as belonging to a cluster.
The procedure start applying a Principal Component Analysis (PCA) keeping the 75% of the original data variance. PCAis not mainly applied to reduce the dimensionality, but to decrease the noise and redistribute the data along new axis (Principal Components) which maximizes the variance and allows the algorithm to better detect the clusters. Then a standardization is applied: this step is crucial, as the algorithm uses Euclidean distances, which are sensitive to the different scales of the variables. Standardizing means transforming the data so that they have a mean of zero and unit variance, allowing all variables to equally influence the distance calculation and clustering.
Next, a multi-dimensional meshgrid (in this case, with 20 dimensions, analogous to the variables

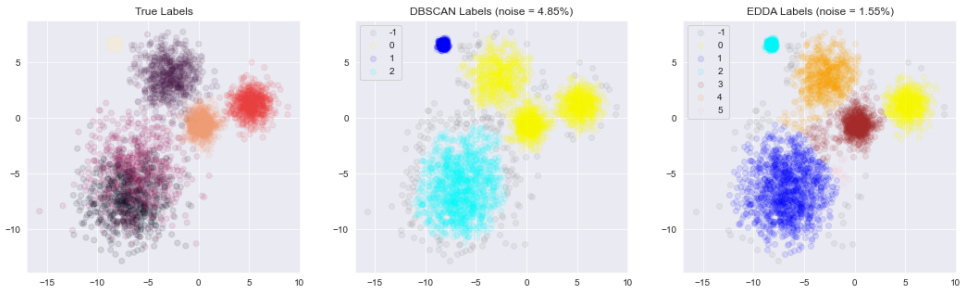


Figure 6.9: Example of EDDA pipeline result compared with classic DBSCAN result. In this simple example 1000 points are sampled by six different bidimensional normal distributions. Each color in the left plot refers to a different distribution, intended as real labels. In the center plot the clusters detected by a classical DBSCAN algorithm are represented, while in the right plot the clusters detected with EDDA pipeline are represented. The comparison shows how the EDDA procedure can better detect smaller (and original) clusters than the classic algorithm.

in each play-vector) is created to estimate a spatial density function in the sample space. The function identifies the local maxima: those samples that have the highest densities. These samples are called **attractors**, and they are selected as such if, in their neighborhood (*vicinity*), they are the samples with the highest spatial density. The radius r, within which the neighborhood is defined, is an input parameter of the pipeline and plays a role similar to that of epsilon in classic DBSCAN. After identifying the attractors, the core part of this approach begins. Here, every sample not defined as an attractor is moved in space based on an attraction law towards the attractor points, similar to a gravitational (linear) force. A parameter g controls the attraction force between a sample p and an attractor A_i as a function of their distance $d(p, A_i) = d_p$. The new positions calculated for each non-attractor sample are simplified and do not follow the real law of gravitation. In fact, the new positions are calculated as:

$$p_i' = p_i + g \cdot d_p,$$

where p_i represents a dimension of sample p. However, more physically consistent versions could be implemented by modifying the attraction law just explained.
Once the samples are processed, they are compacted towards the high-density areas of the original dataset. It is important to find the right g parameter to avoid overshooting the samples away from the attractors: in fact, a value that is too high might pull the samples too strongly, projecting them far away due to the absence of friction in the formula. In my thesis work, a value between

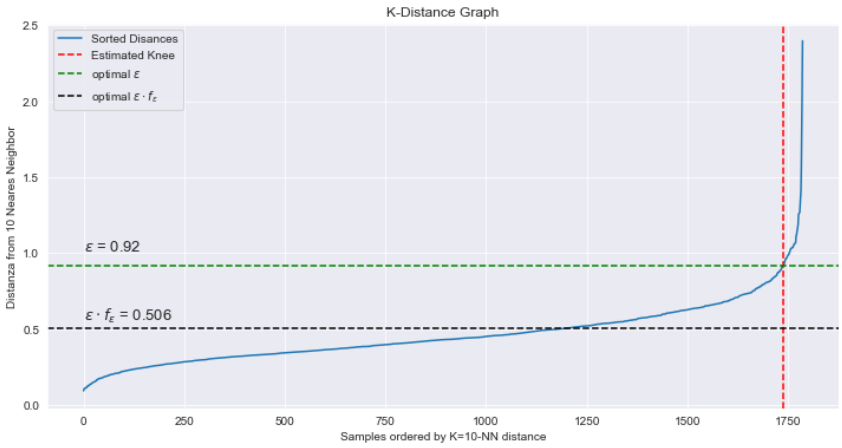


Figure 6.10: Best epsilon computed in first clustering analysis by Elbow tecnique. The epsilon found ($\varepsilon^* = 0.92$) was reduced by a factor $f_\varepsilon = 0.55$, leading to $\tilde{\varepsilon} = 0.506$.

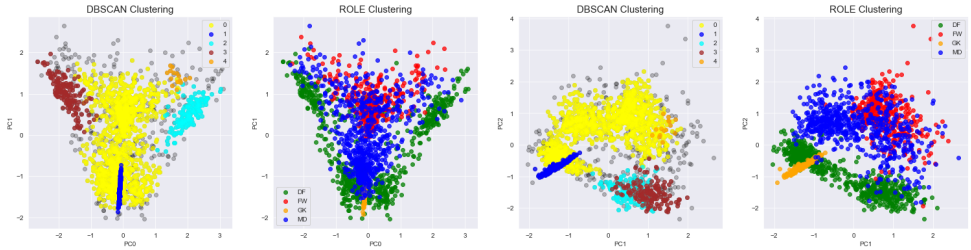


Figure 6.11: Labels obtaind by DBSCAN implemented in first analysis compared with ground truth, the real generic roles.

0.5 and 0.8 proved optimal. After the attraction procedure, DBSCAN is applied as described in the previous subsection: the optimal epsilon ε^* is calculated, reduced by a multiplication factor f_ε, and finally, the clustering algorithm is applied with parameters $\tilde{\varepsilon} = \varepsilon^* \cdot f_\varepsilon$ and an arbitrary $minPts$.
In Figure 6.9, a simple application of the described pipeline can be seen. In the left scatter plot, 5000 simulated samples and their true labels are shown. In the middle and right figures, the resulting labels for classic DBSCAN (middle) and the procedure described in this section (right) are displayed. Two important differences can be observed: the first is the detection of more sub-clusters, which in this case reflect the real clusters; the second is a significant reduction in noise, as attracting the farthest samples decreases the likelihood that they will be spatially excluded from the high-density areas.

6.2.3 Clustering Results

The first analysis conducted was done using the classic DBSCAN. The data was initially standardized and then processed by the algorithm, setting $minPts = 10$, and using the Elbow Technique, $\varepsilon^* = 0.92$ was obtained, reduced to $\tilde{\varepsilon} = 0.506$ with $f_\varepsilon = 0.55$, as shown in Figure 6.10. Figure 6.11 shows a visualization of the data, projected along the three principal components of PCA, comparing the labels obtained from DBSCAN with the real ones, that is, the generic roles: GK (goalkeepers), DF (defenders), MD (midfielders), and FW (forwards). Although the objective of the analysis is not to derive the real original generic roles (but rather to detect sub-patterns in playing styles), the comparison helps us infer the quality of the clusters obtained. As shown by the histograms in Figure 6.12, we can understand which generic roles each cluster contains, excluding the noise which in this case is composed of 15.39% of the samples. We can observe how cluster 1 ($C1$) collected all the goalkeepers, isolating them from other players: this confirms the

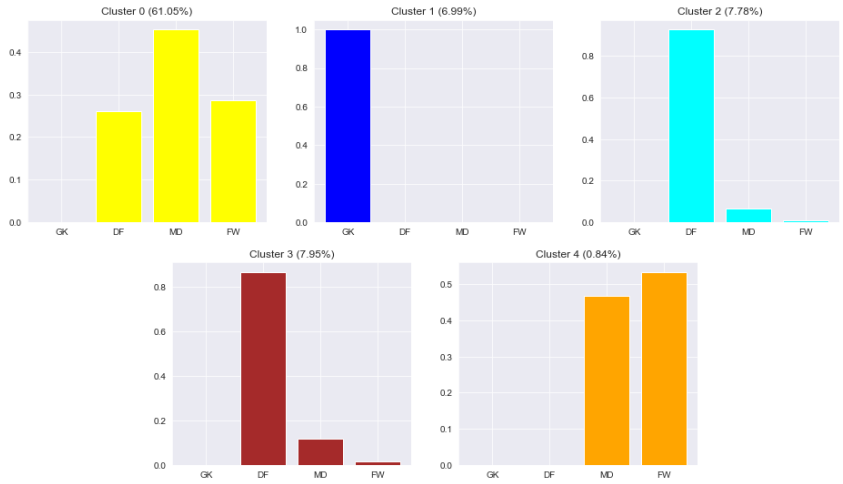


Figure 6.12: Generic roles frequencies in each clusters computed in first analysis.

good representativeness of play-vectors for goalkeepers, who are well distinguished from other roles. We also see how $C2$ and $C3$ collect most of the defenders, although some midfielders are present inside them. This should not necessarily be seen as an error: as we will see shortly, many midfielders have similar playing patterns to some defenders, especially those employed on the wings of the field. $C4$ highlights how most forwards are indistinguishable from many midfielders, as also shown by $C0$, which also includes a fair number of defenders due to the noise present in the original play-vectors dataset.
This analysis is based on the desire to push the algorithm in searching for the smallest playing patterns, and therefore this result did not meet the initial expectations. Thus, goalkeepers were later removed from the original dataset: they are easily isolated in a specific cluster, and by removing them, PCA can spread the remaining samples more "widely" in space, helping to detect sub-patterns.

From the same pipeline, but without the goalkeepers, we obtain $\tilde{\varepsilon} = 0.573$ ($\varepsilon^* = 1.042, f_\varepsilon = 0.55$). In this case the noise is composed by the 20.41% of the total samples. Figure 6.13 shows a comparison between the labels obtained from clustering and the real ones, while Figure 6.14 shows the frequencies of each generic role within each cluster. It can be observed that among defenders, midfielders, and forwards, one more cluster has been detected: this is probably due to a better redistribution by PCA, without the presence of goalkeepers. The additional cluster seems to be the one now labeled as $C2$ (cyan in the figures), and it shows that a new category of defenders is now clearly distinguished. To better understand the detected patterns, it is useful to directly analyze the players' heatmaps. As highlighted in Figure 6.15, the three clusters successfully distinguish three types of defenders: full-backs (left and right) and central defenders. The full-backs, corresponding to clusters $C1$ and $C3$, also contain, as shown in the related histograms, several midfielders: in modern football, many teams adopt a dynamic for-

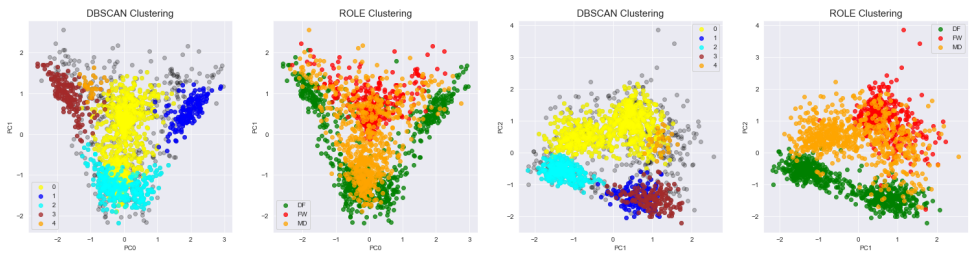


Figure 6.13: Labels obtaind by DBSCAN implemented in second analysis compared with ground truth, the real generic roles.

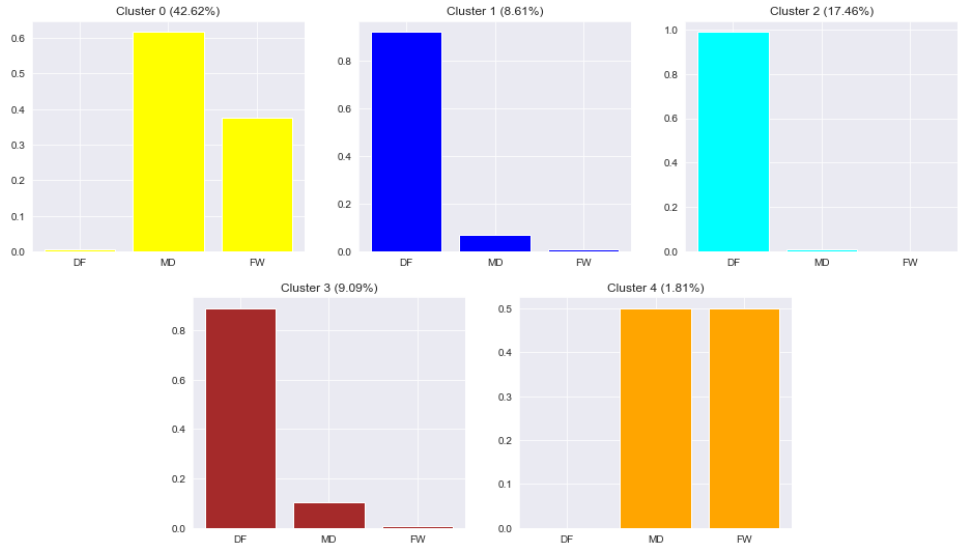


Figure 6.14: Generic roles frequencies in each clusters computed in second analysis.

mation, where roles tend to mix depending on the situation, and it is not uncommon for a wide midfielder to frequently drop back; similarly, it is not rare to see full-backs pushing forward and participating in the offensive play (this is typical, for example, of the 4-2-3-1 formation, which has made players like Marcelo and Carvajal frequent scorers). However, it can still be seen that midfielders and forwards remain indistinguishable. After observing the limitations of the classic DBSCAN, the EDDA pipeline was implemented. PCA was applied to the data (keeping 75% of the original variance), and the data was standardized. Then, following the procedure described in the previous subsection, the attractor samples were obtained using $r = 1$ and $g = 0.75$. The figure 6.16 shows the result of this step, plotting the samples from three different PCA point of view. The red points are the attractor samples: can be clearly seen that actractors aren't all representative of the same density, in fact the "maximum density" computated is relative in space. This fact allows to retrieve cluster of different densities and sizes, helping the playing sub-patterns detection. Gray samples represent the original data, while the blu ones represent the new positions computer after the attraction. From the plots can be easily seen how the improvement works: sub-clusters are much more visible in new data compared with original data.
Lately, the DBSCAN algorithm was implemented using the previous analysis parameters. How is shown in figure 6.17, the improvement is clearly visible. I got a total of 13 clusters with a very low percentage of noise: 8.97% of total samples was considerated outliers (11.44% less than the previous case). Before getting deeper into the analysis of each cluster heatmaps, is interesting to observe the real roles histograms, shown in figure 6.18. Now six clusters include the defenders ($C1$, $C3$, $C6$, $C7$, $C9$, and $C10$), with only two ($C6$ and $C9$) showing a modest contribution from midfielders. As can be seen in the appendix, $C6$ and $C9$ represent the right and left full-backs, respectively. This role requires covering the entire side of the field, with defensive duties under pressure and offensive contributions such as crosses, shots, and link-ups with the midfield. This last characteristic can be clearly seen from the spatial densities of pass endings, mostly concentrated between the corresponding wing and the central area of the field. The remaining four clusters represent the main pure defensive roles: central defenders (right and left) and full-backs. It is interesting how the algorithm detected a specific sub-pattern represented by the samples in cluster $C10$: this cluster is dedicated only to those central defenders in an odd-numbered defensive line (such as a three or five-man defense) whose job is to cover behind the two wide defenders. This position is often referred to as the "sweeper".
The midfielders are mainly divided into five clusters: $C0$, $C4$, $C5$, $C8$, and $C11$. $C0$ and $C8$ are the only clusters made up of only midfielders and seem to include those involved in the lower-

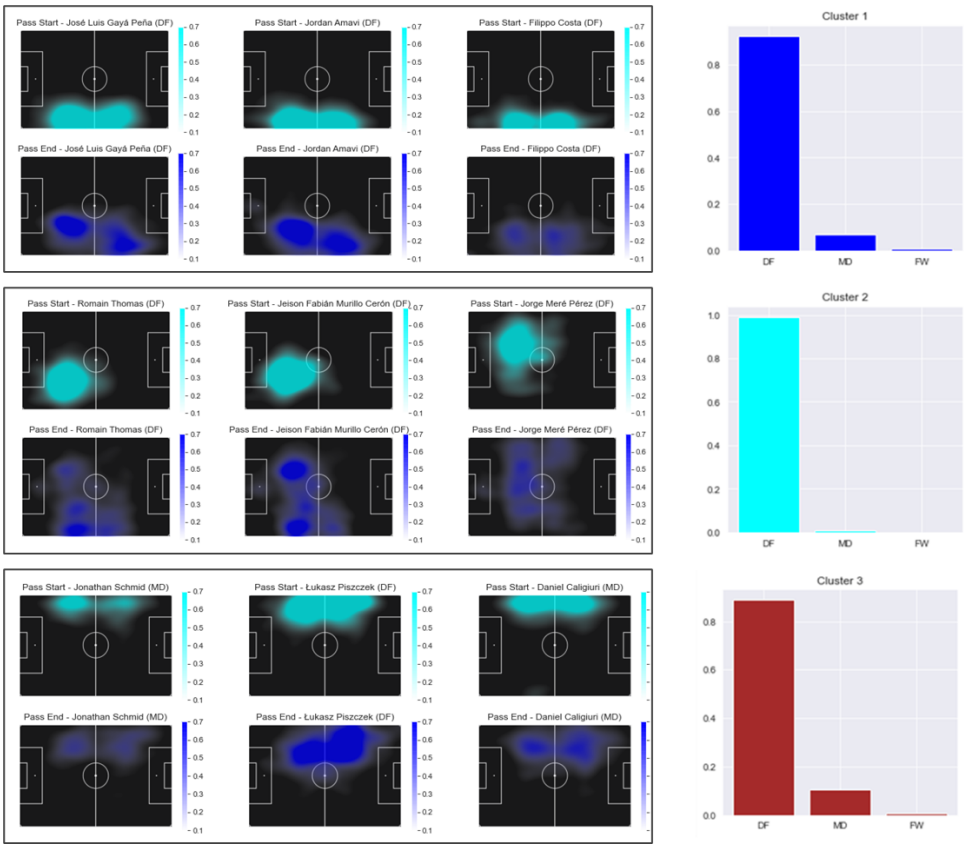


Figure 6.15: The *Pass Start* and *Pass End* heatmaps are shown for the clusters $C1$, $C2$ and $C3$. The three clusters show distinguishable patterns, related to full-backs ($C3$ left and $C1$ right) and central defenders ($C2$).

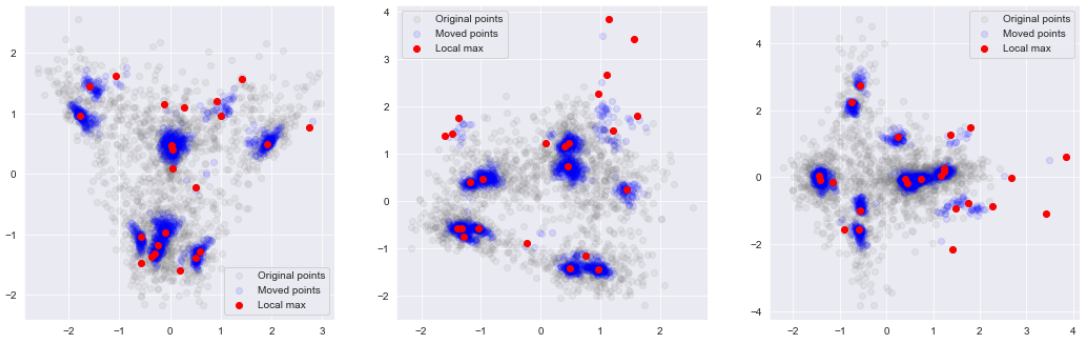


Figure 6.16: Schematic representation of the attraction mechanism in the EDDA pipeline. The original samples (in gray) are pushed to the attractors (in red) with a "gravity" factor g, in function of their distance with each attractor. The new sample positions are represented in blue, and can be seen how close to the actractors they are.

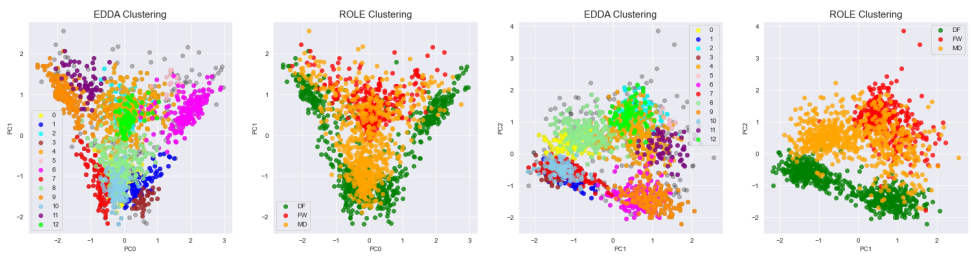


Figure 6.17: Labels obtaind by EDDA pipeline implemented in third analysis compared with ground truth, the real generic roles.

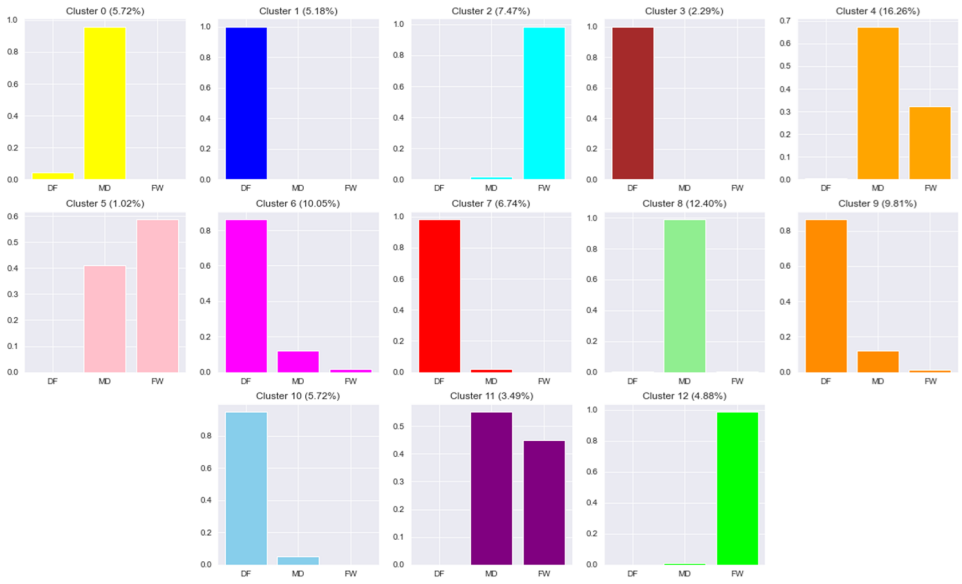


Figure 6.18: Generic roles frequencies in each clusters computed by EDDA pipeline in third analysis.

middle part of the field. These are midfielders focused on ball recovery and playmaking during offensive actions. An example of this category are the "defensive midfielders." It is interesting to note how these two clusters are differentiated by one detail, visible in the heatmaps: $C0$ appears to be characterized by players with higher shooting frequencies, mainly from outside the penalty area. Clusters $C5$ and $C11$ represent typical playing patterns of wingers, on the right and left, respectively. These two also include a significant number of forwards. Cluster $C4$ is the least interpretable of all. In the corresponding heatmaps, a strong presence of dribbling, crosses, and shots can be observed. Probably, considering the strong presence of forwards, the pattern (or mix of patterns) in this cluster involves second strikers, wide midfielders, and attacking midfielders: all players who actively participate in the offensive play, moving around the penalty area.
Finally, the pure forwards seem to be divided into two clusters that completely separate them from the other players: $C2$ and $C12$. While the second one ($C12$) seems to include all pure central strikers, the first one ($C2$) appears to be characterized by mobile forwards, agile attackers who can be dangerous in counterattacks by dribbling past defenders, and target men, who not only score but also hold up the ball for their teammates and free them for a shot.
In summary, we can say that the analyzed clusters show a clear distinction between roles, not only managing to differentiate between defensive, transitional, and offensive roles, but also grouping together samples that, despite having different roles, display the same playing characteristics. The clustering has identified players with specific playing styles: central defenders, full-backs, defensive midfielders, deep-lying playmakers, attacking wingers, and both finishing and mobile forwards. The segmentation accurately reflects the playing patterns.

Conclusions

The goal of my thesis was to develop a tool capable of representing a soccer player's playing style in a compact form, based on their actions on the field. To do this, play-vectors were generated: numerical vectors that project the heatmaps of different types of actions into a relatively small space for each player.

In the first phase, I analyzed and processed a dataset of spatio-temporal events, which collects all actions from matches in the five major European leagues, from the 2017/2018 season. Six different types of actions (or features) were selected, based on specific criteria: pass start, pass end, crosses, shots, dribbles, and tackles. A total of 1787 players were selected out of the initial 3074, considering only those with enough playing time to ensure that the frequency of their actions was representative of their playing style. For each player, six heatmaps were generated, one for each feature, following a precise process: collecting the actions on a spatial grid, normalizing based on the minutes played, smoothing the frequencies with a Gaussian filter, and applying Min-Max scaling to distribute the heatmap values in the range $[0, 1]$.

The heatmaps were used as input for training a Variational Autoencoder and for solving a Non-Negative Matrix Factorization: both techniques were applied to reduce the high dimensionality of the heatmaps into a lower-dimensional space, the play-vectors space. Both techniques successfully produced 20-element vectors.

Inspired by the work of Decroos et al. [3] and Hyeonah et al. [12], I evaluated the effectiveness of the play-vectors in representing playing styles through a *Player Retrieval Analysis*. In this analysis, the play-vectors of each player were generated based on their heatmaps for the first and second halves of each match. Then, I tried to infer the players from their second-half play-vectors by measuring the similarity (using a numerical distance) with the first-half play-vectors. The analysis showed that the vectors obtained from NMF were generally more effective, especially when using cosine distance to quantify the similarity between vectors. These vectors not only demonstrated greater accuracy but, being the result of NMF, offered a crucial advantage: human interpretability. Each value in the NMF-play-vectors represents the importance of a specific latent component, which corresponds to a recurring playing behavior in the dataset, making it easier to analyze and understand tactical patterns.

The vectors obtained from NMF were then used to implement a DBSCAN clustering to investigate the possibility of detecting specific playing patterns among players from the play-vectors. Initially, a standard DBSCAN was implemented, using the elbow technique to help choose *epsilon*. This approach proved barely sufficient: the algorithm detected only five clusters, representing very general playing styles, and estimated 20.41% of the samples as outliers.

Next, I developed an innovative pipeline to preprocess the data before applying DBSCAN, with the goal of reducing noise and helping the algorithm detect smaller and more specific clusters: the Enhanced DBSCAN with Density Attractors pipeline. The mechanism is based on recognizing attractors, samples characterized by high spatial density, and adjusting the positions of other samples based on a kind of "gravity law." After modifying the positions, DBSCAN is applied to

label the samples.

This approach showed improvement in detecting more specific patterns: thirteen different clusters were identified, including six defensive patterns, five characteristic midfield patterns (both defensive and offensive), and two specific for attack, reducing the percentage of outliers to 8.97% of the total samples.

In summary, my thesis developed a tool that allows the analysis of players' playing styles by quantifying the frequency of their actions and compacting them into a vector, which can be used to monitor a player's performance or classify them within a specific playing pattern for market evaluations.

Clusters Visualization

In the following appendix, the heatmaps related to the main features considered in this study are presented for some players labeled in each cluster. The frequencies of each action are specified by the colorbar on the right of each heatmap. Each feature, for each cluster, has the same minimum and maximum values on the colorbar, to facilitate comparison among the selected samples. We also show the density distributions of defenders, midfielders, and forwards within the same cluster, in order to report the role composition of each cluster. This allows for a better understanding of the actual playing style represented by the clusters.

Cluster 0

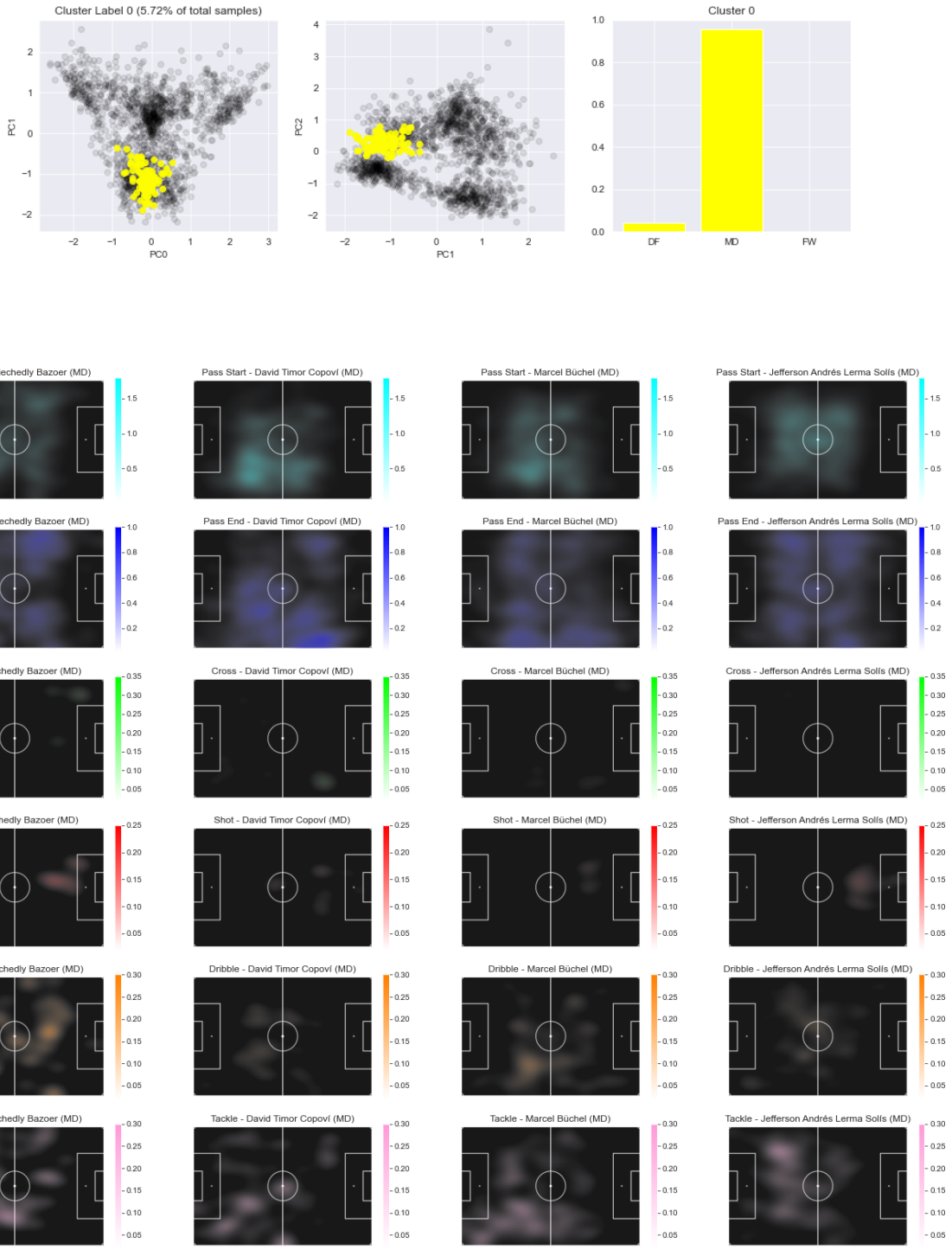
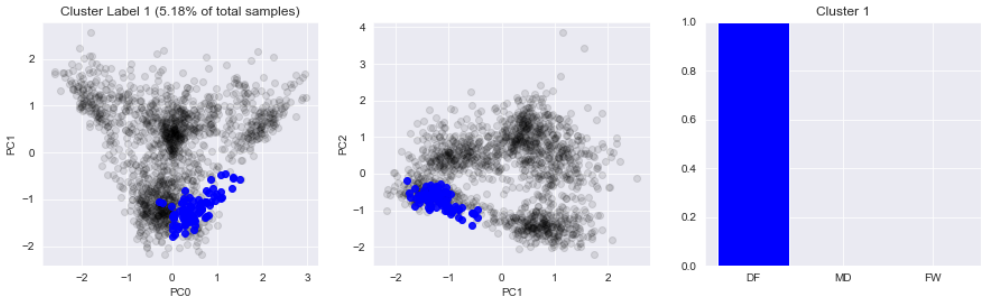


Figure 6.19: Cluster 0.

Cluster 1



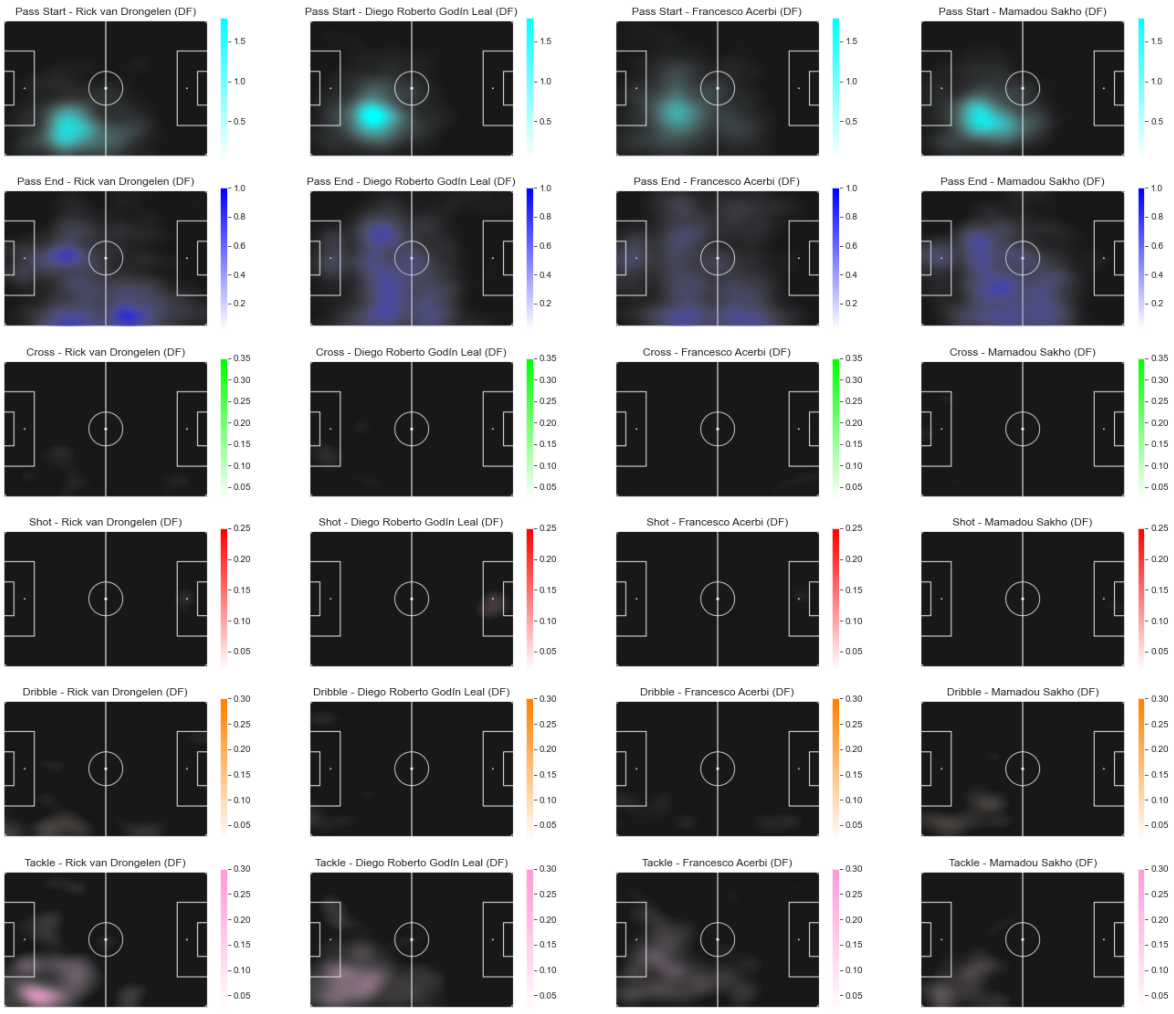
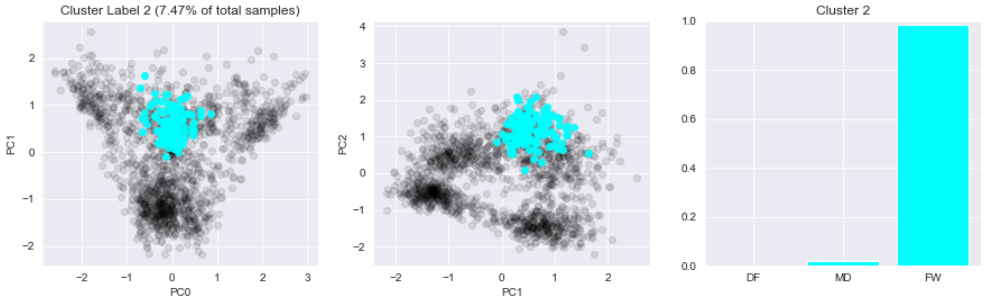


Figure 6.20: Cluster 1.

Cluster 2



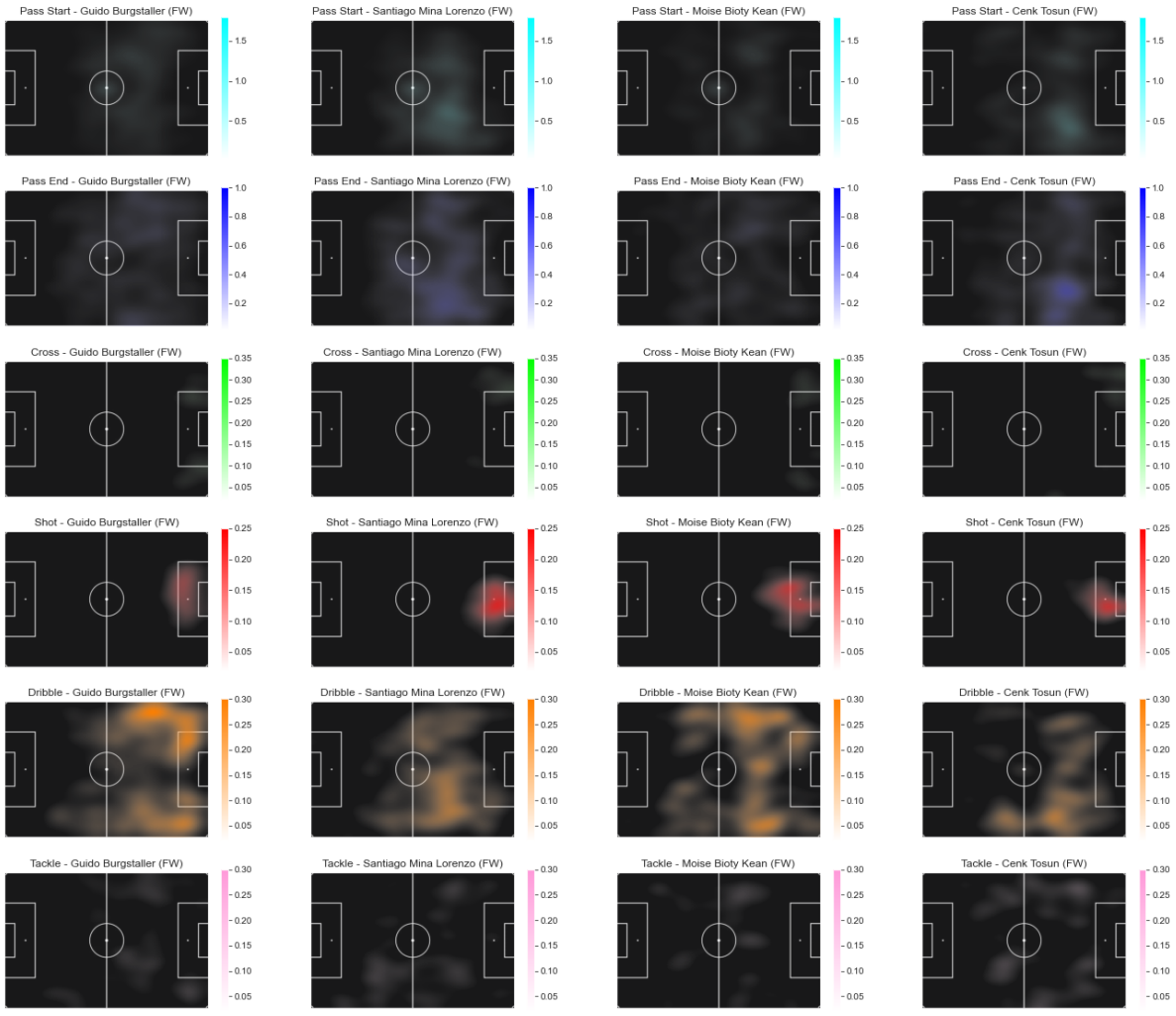
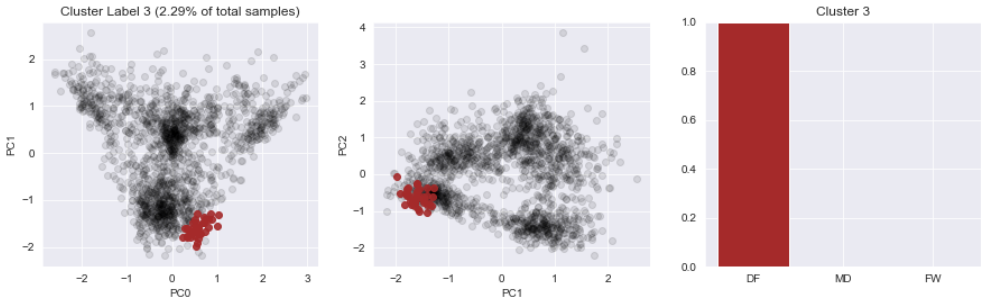


Figure 6.21: Cluster 2.

Cluster 3



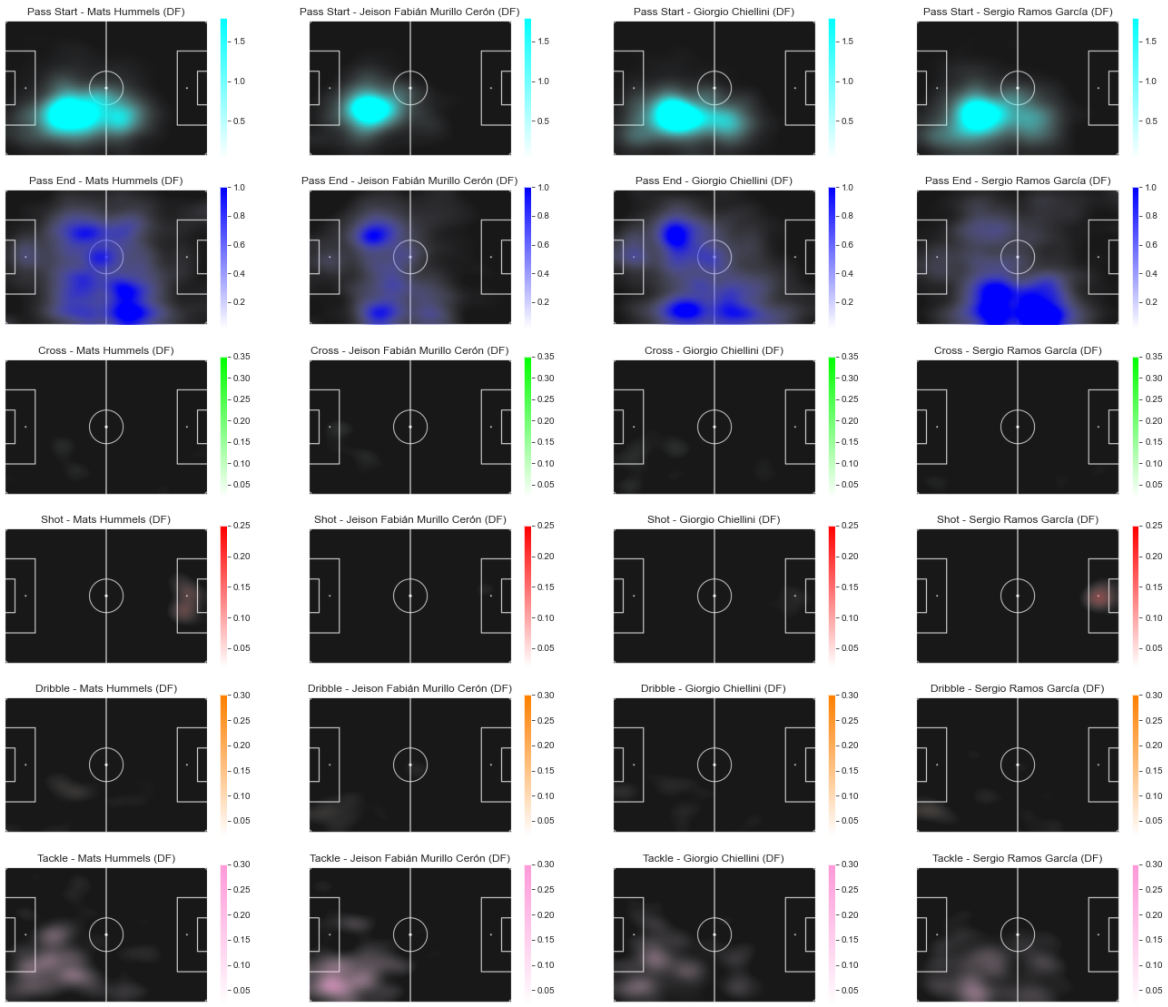
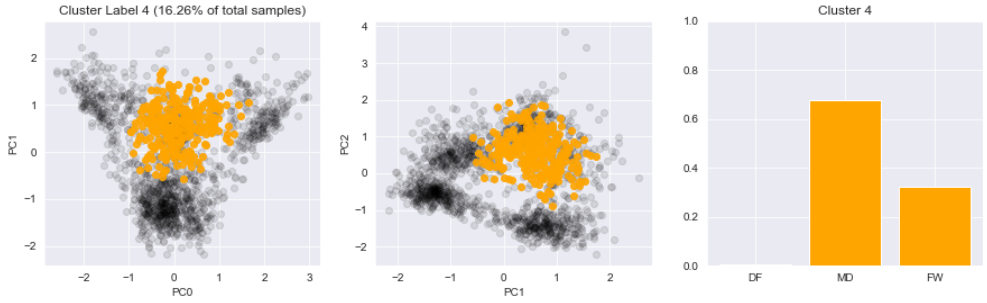


Figure 6.22: Cluster 3.

Cluster 4



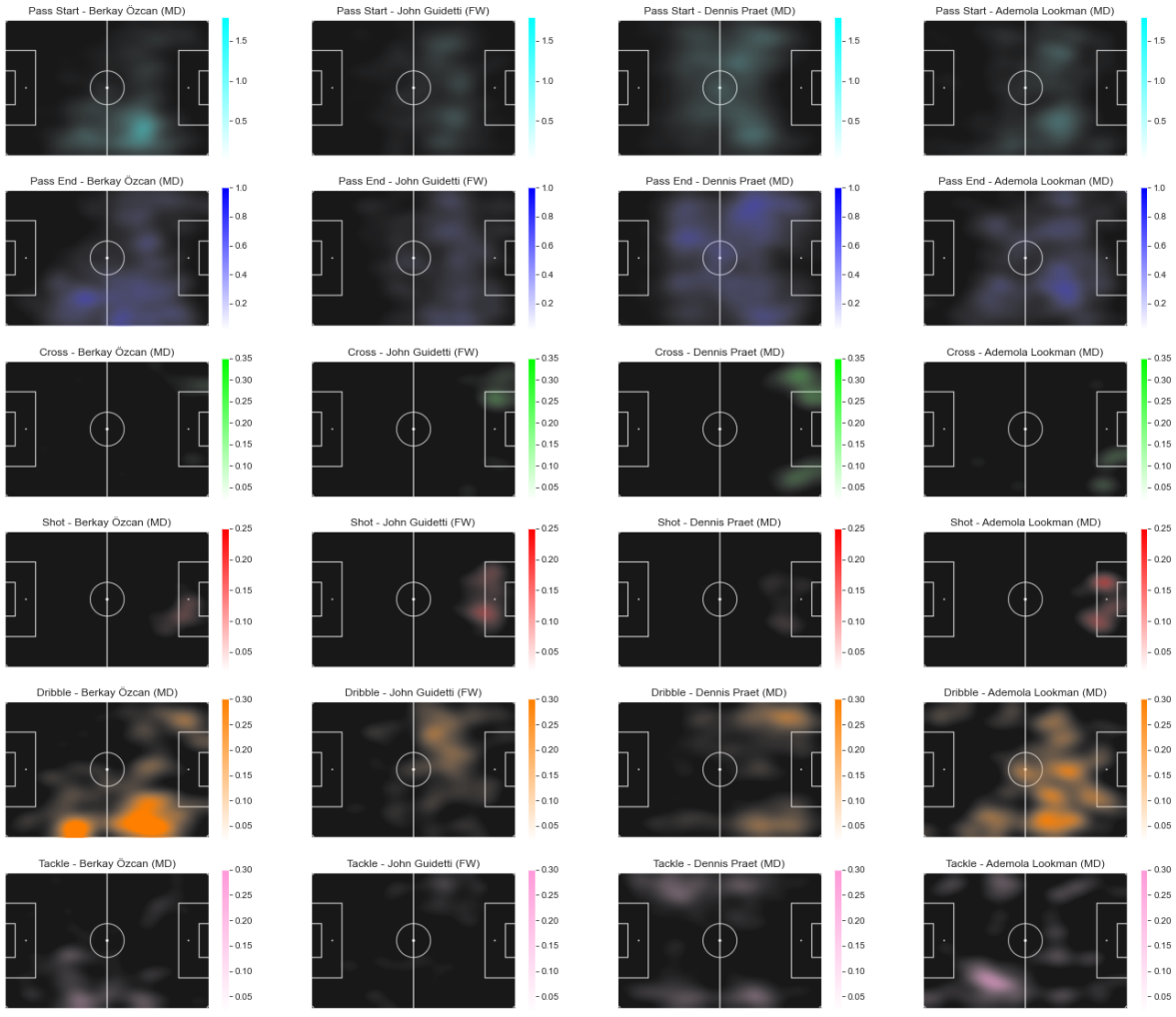
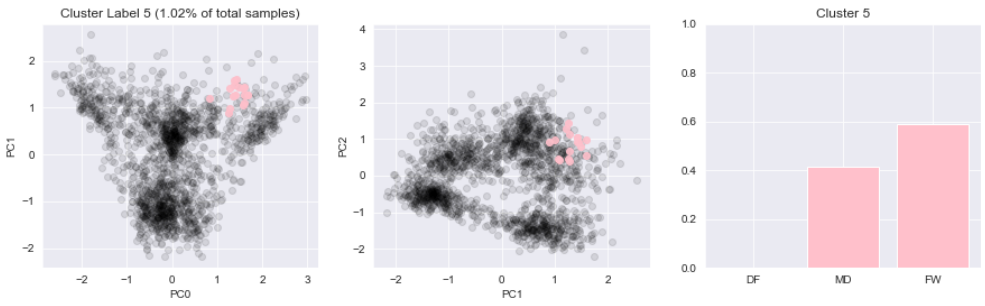


Figure 6.23: Cluster 4.

Cluster 5



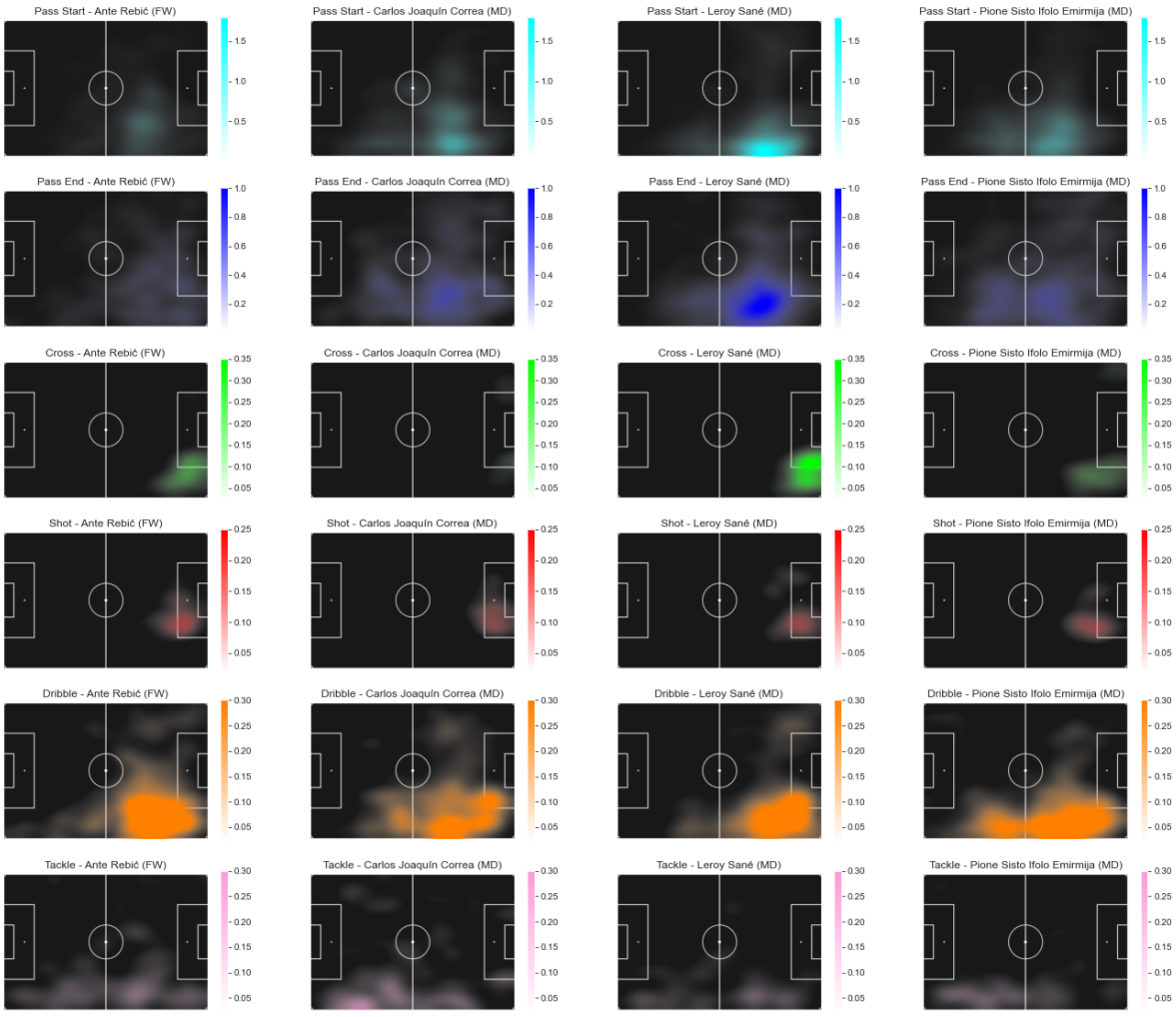
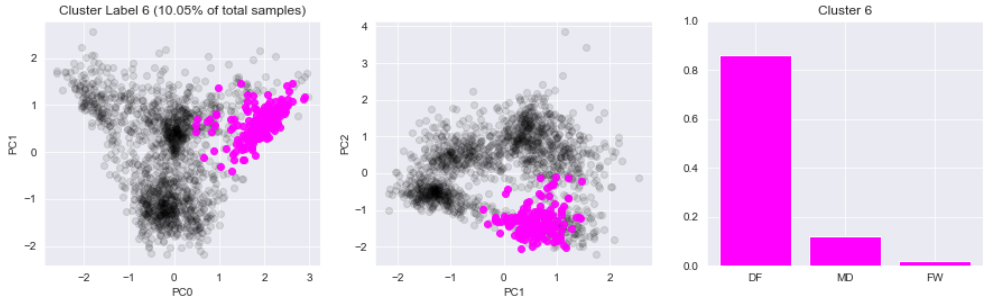


Figure 6.24: Cluster 5.

Cluster 6



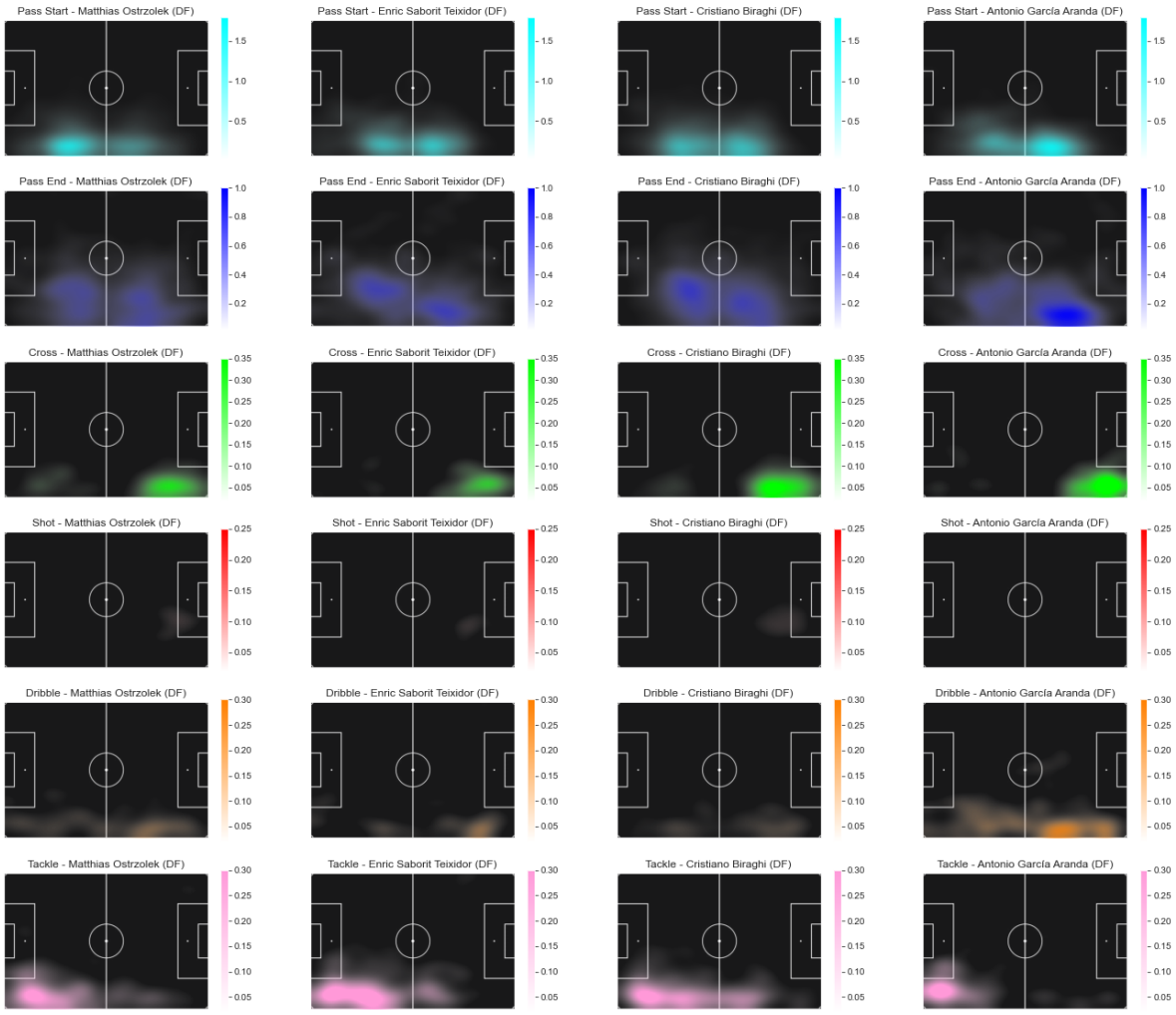
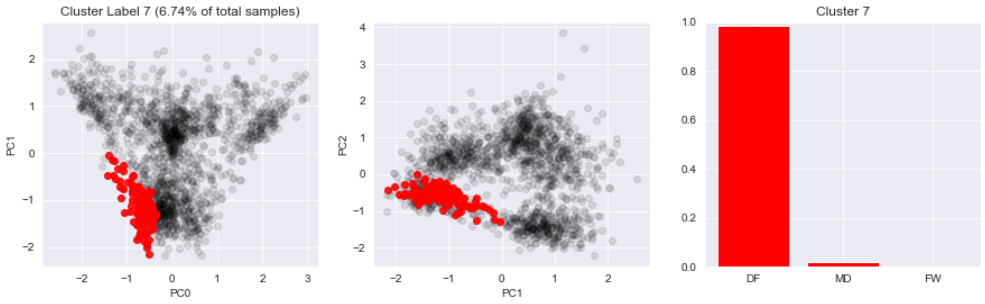


Figure 6.25: Cluster 6.

Cluster 7



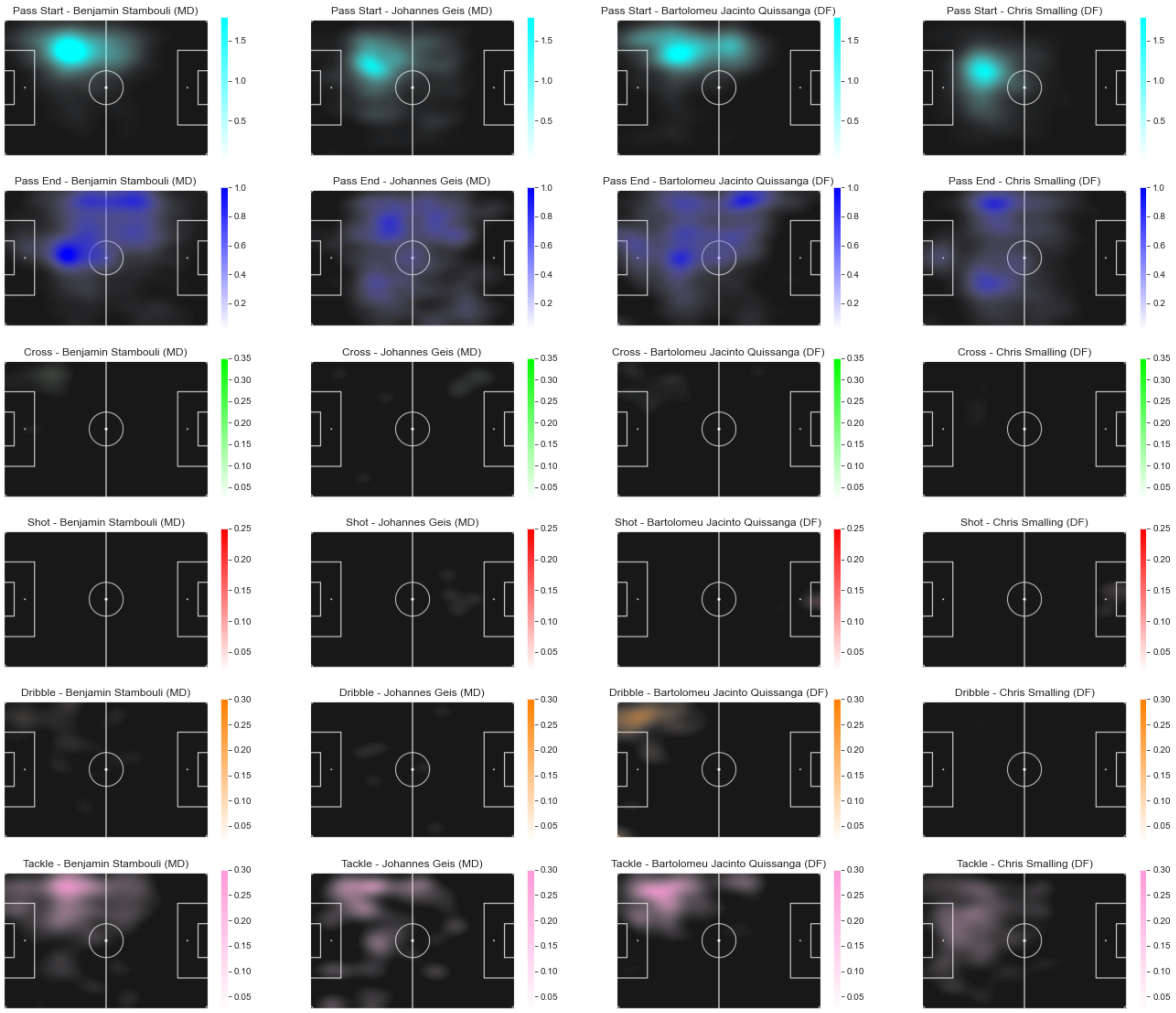
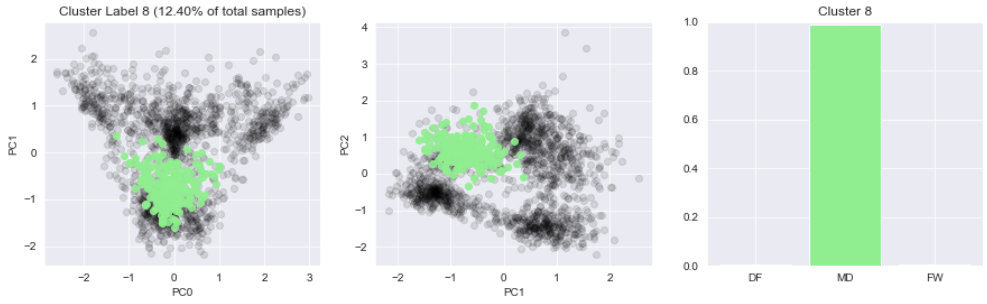


Figure 6.26: Cluster 7.

Cluster 8



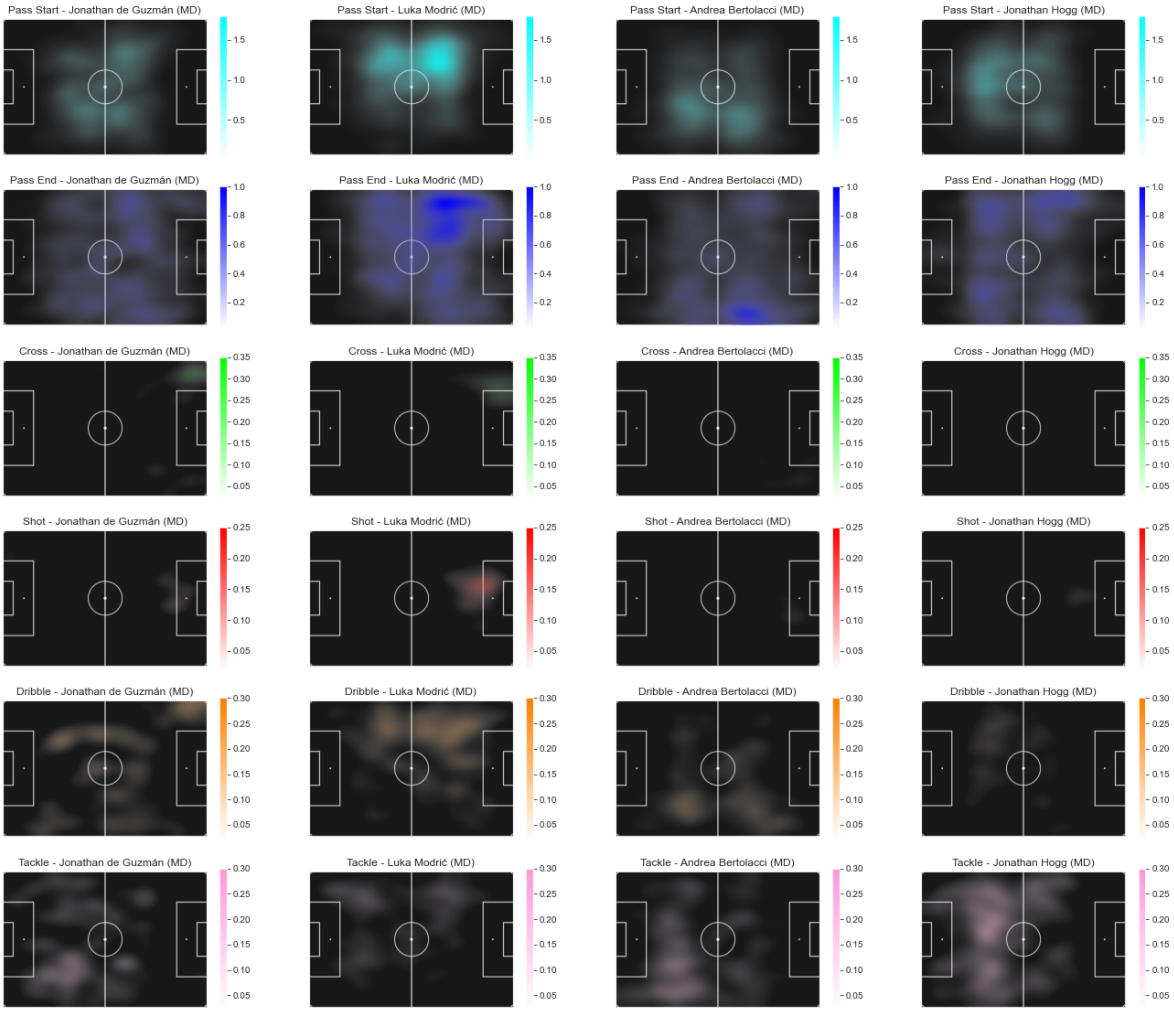
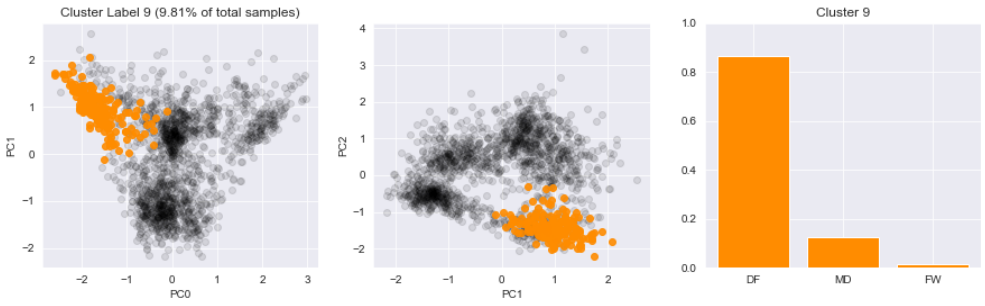


Figure 6.27: Cluster 8.

Cluster 9



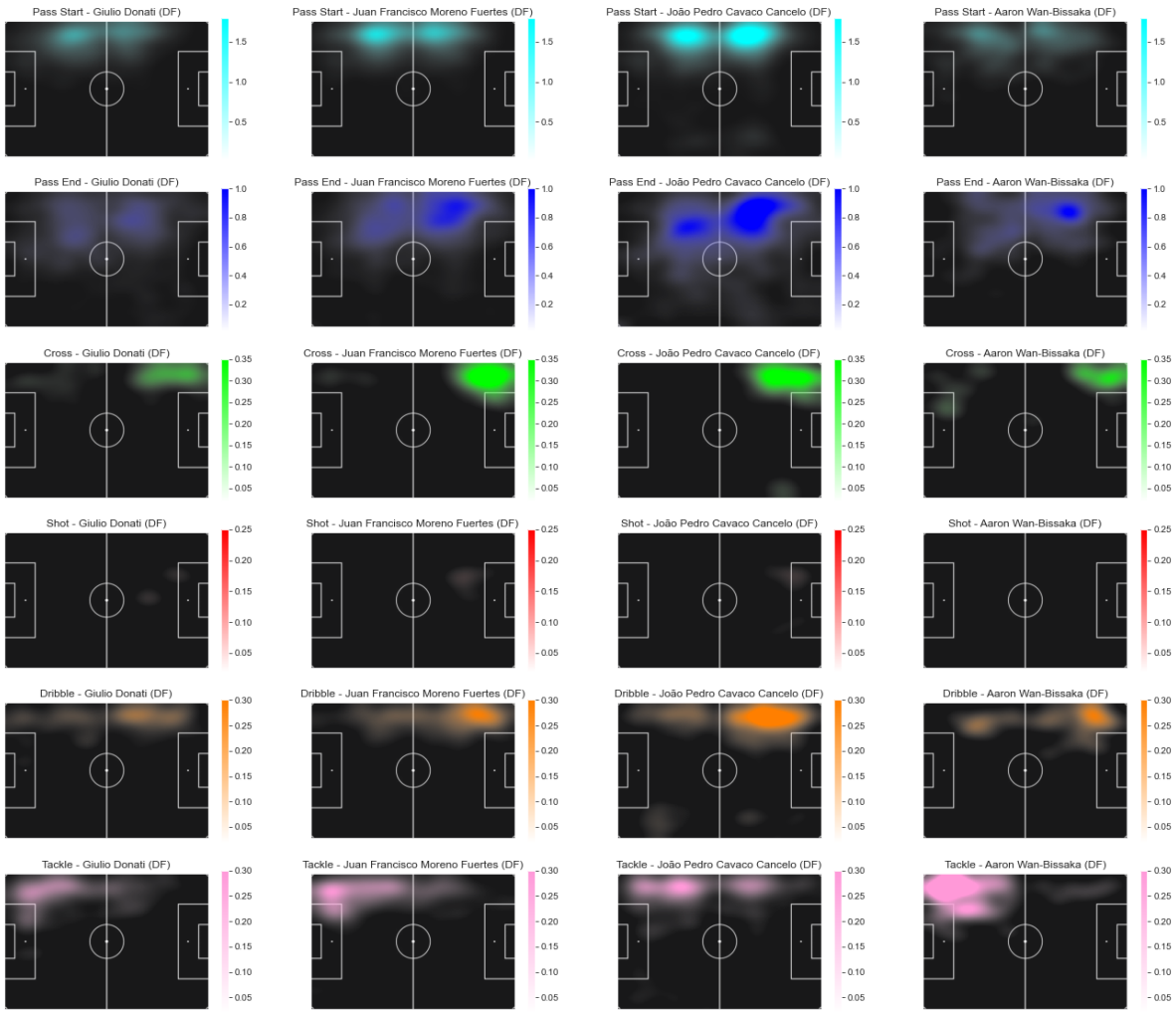


Figure 6.28: Cluster 9.

Cluster 10

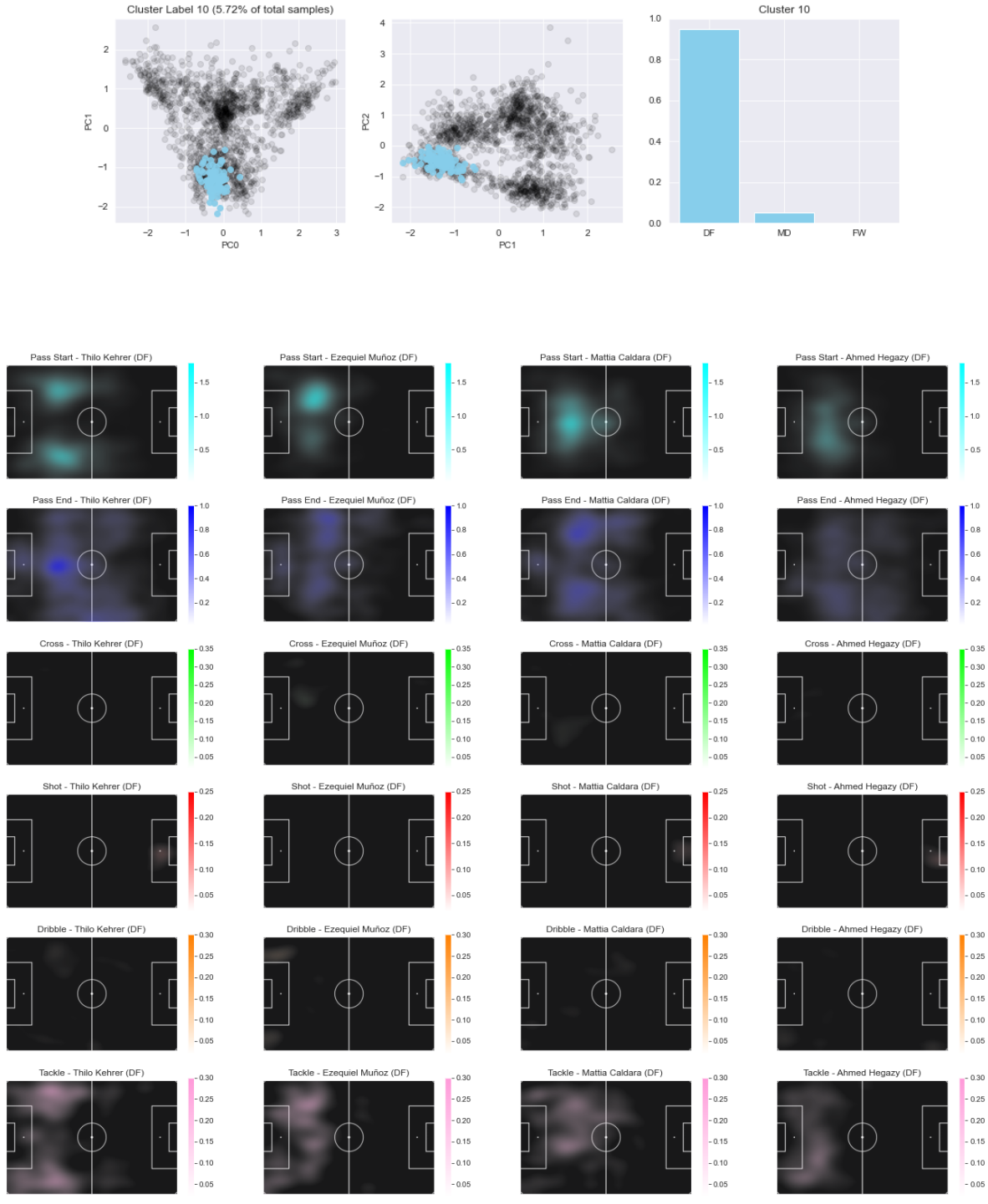
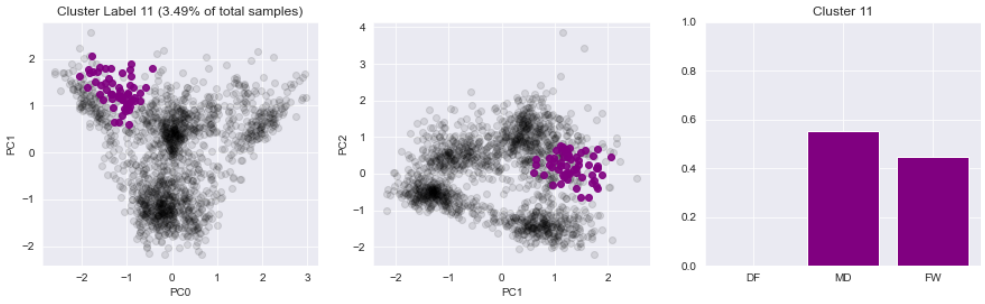


Figure 6.29: Cluster 10.

Cluster 11



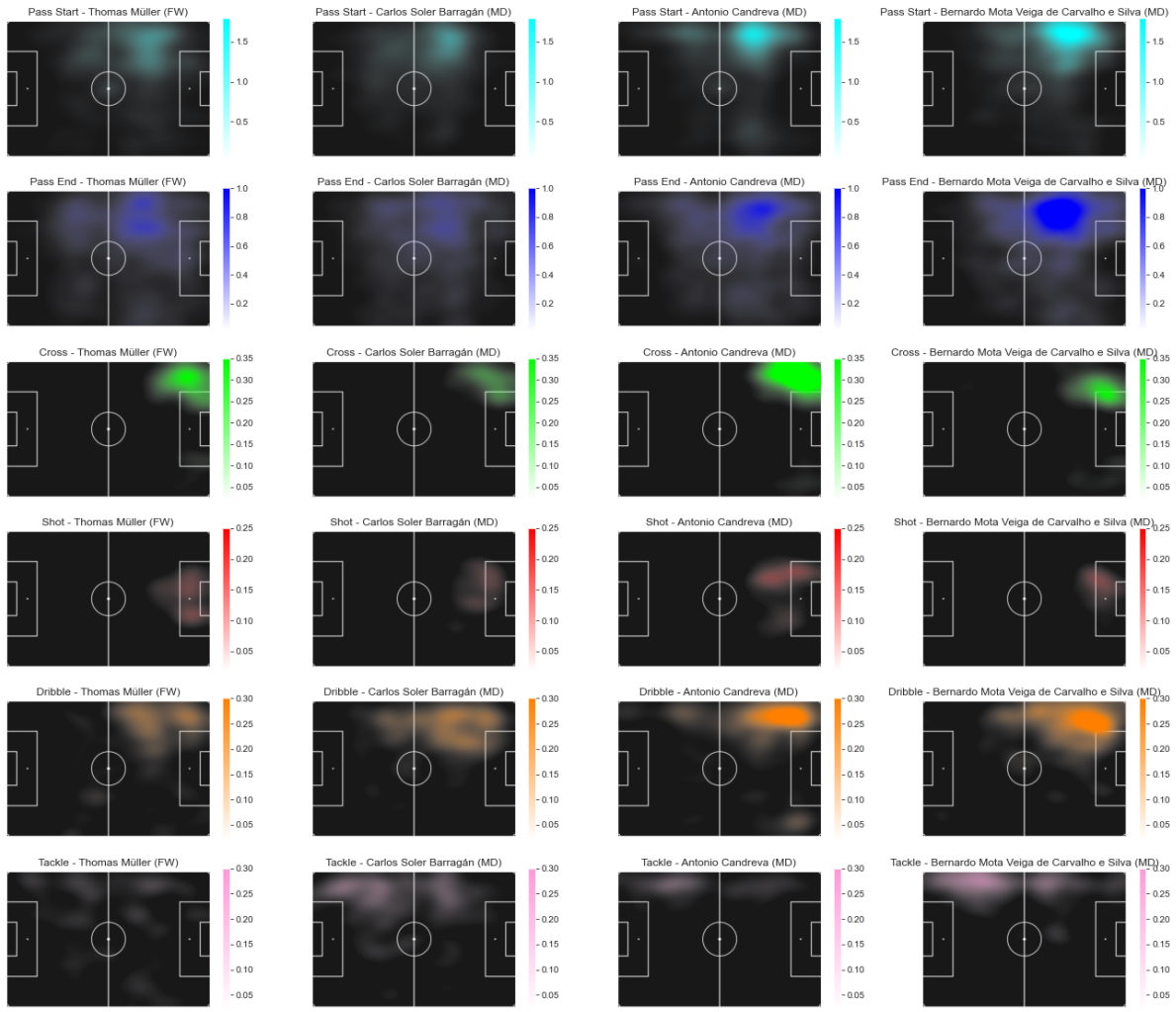
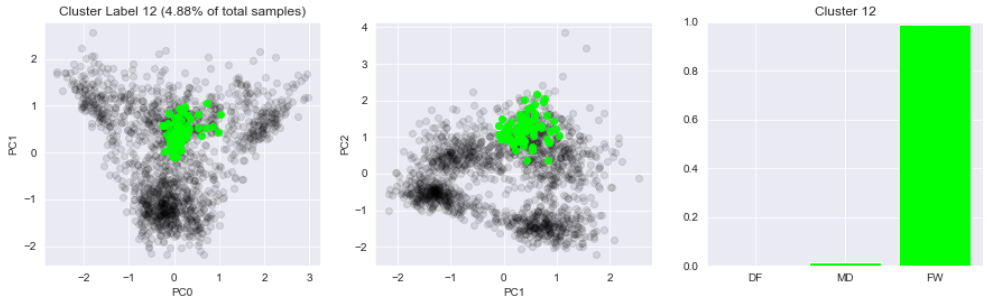


Figure 6.30: Cluster 11.

Cluster 12



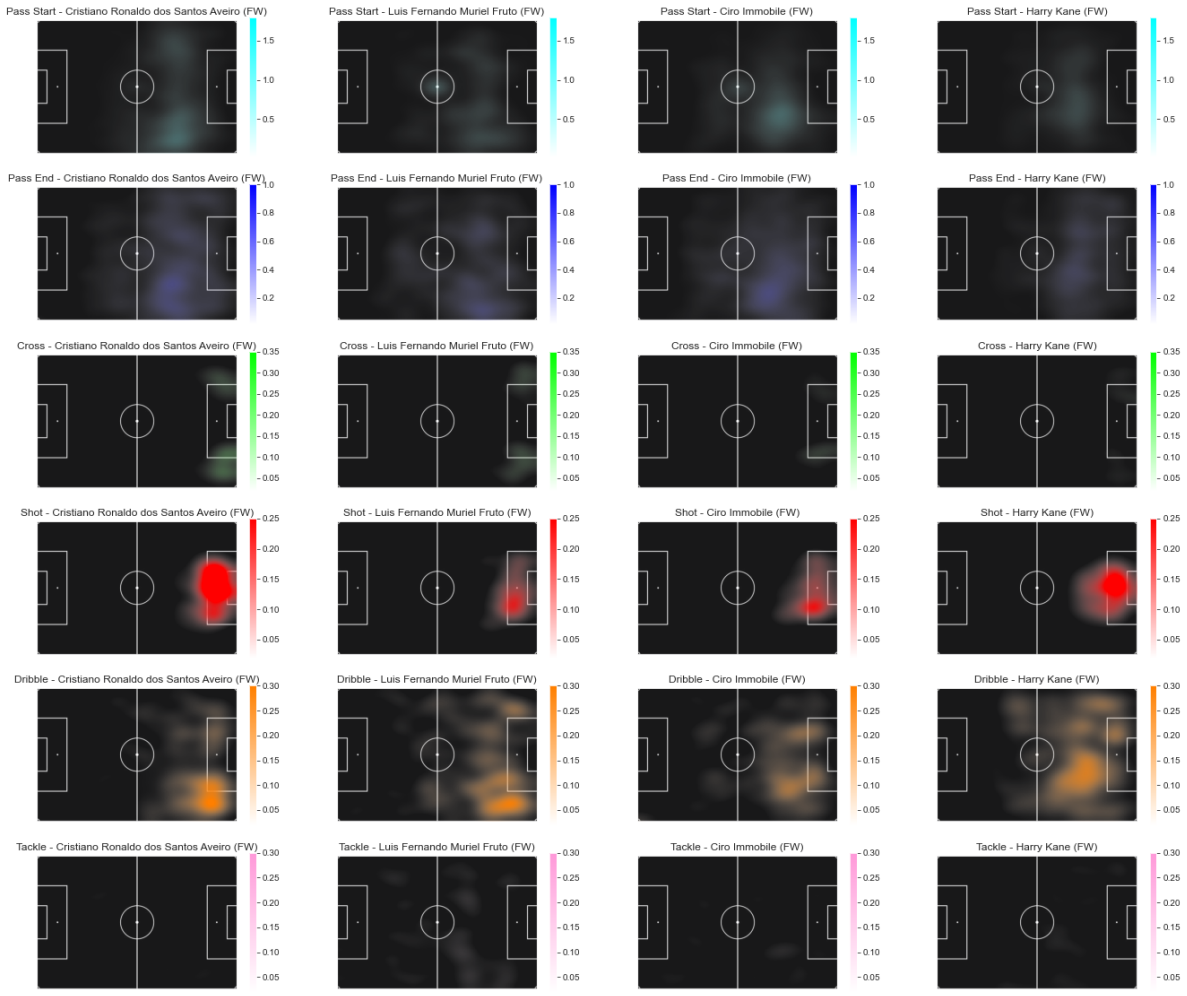


Figure 6.31: Cluster 12.

Bibliography

[1] Bruce Reider. "Moneyball". In: *The American Journal of Sports Medicine* 42.3 (2014), pp. 533–535.

[2] Luca Pappalardo et al. "A public data set of spatio-temporal match events in soccer competitions". In: *Scientific data* 6.1 (2019), p. 236.

[3] Tom Decroos and Jesse Davis. "Player vectors: Characterizing soccer players' playing style from match event streams". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III*. Springer. 2020, pp. 569–584.

[4] David L Donoho et al. "High-dimensional data analysis: The curses and blessings of dimensionality". In: *AMS math challenges lecture* 1.2000 (2000), p. 32.

[5] D Morgenstern and Richard Bellman. "Adaptive control processes: a guided tour". In: *Econometrica* 30.3 (1962), p. 599.

[6] Andrea Bommert et al. "Benchmark for filter methods for feature selection in high-dimensional classification data". In: *Computational Statistics & Data Analysis* 143 (2020), p. 106839.

[7] B Azhagusundari, Antony Selvadoss Thanamani, et al. "Feature selection based on information gain". In: *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* 2.2 (2013), pp. 18–21.

[8] Agustin Garcia Asuero, Ana Sayago, and AG González. "The correlation coefficient: An overview". In: *Critical reviews in analytical chemistry* 36.1 (2006), pp. 41–59.

[9] Takio Kurita. "Principal component analysis (PCA)". In: *Computer vision: a reference guide* (2019), pp. 1–4.

[10] Petros Xanthopoulos et al. "Linear discriminant analysis". In: *Robust data mining* (2013), pp. 27–33.

[11] Michael Tschannen, Olivier Bachem, and Mario Lucic. "Recent advances in autoencoder-based representation learning". In: *arXiv preprint arXiv:1812.05069* (2018).

[12] H. Cho, H. Ryu, and M. Song. "Pass2vec: Analyzing soccer players' passing style using deep learning". In: *International Journal of Sports Science and Coaching* 17.2 (2022), pp. 355–365.

[13] Juan Camilo Campos. "Determining the phases of play using Graph Neural Network Embeddings". In: ().

[14] Pentti Paatero and Unto Tapper. "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values". In: *Environmetrics* 5.2 (1994), pp. 111–126.

[15] Daniel D Lee and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *nature* 401.6755 (1999), pp. 788–791.

[16] Robert Gray. "Vector quantization". In: *IEEE Assp Magazine* 1.2 (1984), pp. 4–29.

[17] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *Machine learning for data science handbook: data mining and knowledge discovery handbook* (2023), pp. 353–374.

[18] Shuangshuang Chen and Wei Guo. "Auto-encoders in deep learning—a review with new perspectives". In: *Mathematics* 11.8 (2023), p. 1777.

[19] Hanyu Xiang et al. "Deep learning for image inpainting: A survey". In: *Pattern Recognition* 134 (2023), p. 109046.

[20] Lucas Pinheiro Cinelli et al. "Variational autoencoder". In: *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer, 2021, pp. 111–149.

[21] Raymond W Yeung. *Information theory and network coding*. Springer Science & Business Media, 2008.

[22] John T McCoy, Steve Kroon, and Lidia Auret. "Variational autoencoders for missing data imputation with application to a simulated milling circuit". In: *IFAC-PapersOnLine* 51.21 (2018), pp. 141–146.

[23] Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[24] Hudl. *Wyscout: The world's biggest library of football video and data*. Accessed: 2024-09-05.

[25] Luca Pappalardo and Emanuele Massucco. *Soccer match event dataset*. Collection. 2019.

[26] FBref. *FBref.com - Statistiche sul calcio e storia*. Accessed: 03/09/2024.

[27] Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*. Vol. 17. Cambridge university press, 2023.

[28] SciPy Developers. *scipy.ndimage.gaussian_filter*. Accessed: 2024-09-06. 2024.

[29] Vinod Sharma. "A study on data scaling methods for machine learning". In: *International Journal for Global Academic & Scientific Research* 1.1 (2022), pp. 31–42.

[30] Christos Boutsidis and Efstratios Gallopoulos. "SVD based initialization: A head start for nonnegative matrix factorization". In: *Pattern recognition* 41.4 (2008), pp. 1350–1362.

[31] Hervé Abdi. "Singular value decomposition (SVD) and generalized singular value decomposition". In: *Encyclopedia of measurement and statistics* 907.912 (2007), p. 44.

[32] E. Guven. "Decision of the Optimal Rank of a Nonnegative Matrix Factorization Model for Gene Expression Data Sets Utilizing the Unit Invariant Knee Method: Development and Evaluation of the Elbow Method for Rank Selection". In: *JMIR Bioinformatics and Biotechnology* (2023).

[33] Yun Cai, Hong Gu, and Toby Kenney. "Rank selection for non-negative matrix factorization". In: *Statistics in Medicine* 42.30 (2023), pp. 5676–5693.

[34] Wei Luo et al. "Convolutional sparse autoencoders for image classification". In: *IEEE transactions on neural networks and learning systems* 29.7 (2017), pp. 3289–3294.

[35] Munmi Gogoi and Shahin Ara Begum. "Image classification using deep autoencoders". In: *2017 IEEE international conference on computational intelligence and computing research (ICCIC)*. 2017.

[36] Fabian Duffhauss et al. "FusionVAE: A deep hierarchical variational autoencoder for RGB image fusion". In: *European Conference on Computer Vision*. Springer. 2022, pp. 674–691.

[37] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[38] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[39] RL Thorndike. *Who belongs in the family? Pyschometrika 18 (4): 267–276*. 1953.

[40] Kevin Arvai. *kneed (v0.8.5)*. 2023. DOI: 10.5281/zenodo.8127224. URL: https://doi.org/10.5281/zenodo.8127224.