Contrasting Learning Pathways: Supervised and Unsupervised Analysis on Fifa 23 Player Data

2023-09-13

Intro

FIFA 23, like its predecessors, offers a goldmine of player data that can be employed to extract insights, predict outcomes, and generally push the boundaries of football analytics. In this article, we're going to juxtapose two broad approaches to machine learning using FIFA 23's player data: Supervised Learning and Unsupervised Learning. We shall be predicting/clustering player positions based on information such as player height, weight, salary etc.

Setting Up

We start by loading the necessary libraries and reading the dataset. Our dataset contains various attributes for FIFA 23 players such as their positions, overall ratings, age, preferred foot, and other skills ratings.

```
library(FNN)
library(caret)
library(randomForest)
library(e1071)
library(ggplot2)
library(class)
library(readr)
library(missForest)
library(uwot)
library(NMF)
library(Rtsne)
library(dplyr)
```

Data Preparation

Loading and Exploring the Player Data

```
data <- read_csv("/Users/muhammadluay/Library/Containers/com.microsoft.Excel/Data/Desktop/random documen
head(data)
```

```
## # A tibble: 6 x 110
##   sofifa_id player_url   short_name long_name player_positions overall potential
##       <dbl> <chr>        <chr>      <chr>     <chr>              <dbl>     <dbl>
## 1    158023 https://sof~ L. Messi   Lionel A~ RW, ST, CF            93        93
## 2    188545 https://sof~ R. Lewand~ Robert L~ ST                    92        92
## 3     20801 https://sof~ Cristiano~ Cristian~ ST, LW                91        91
## 4    190871 https://sof~ Neymar Jr  Neymar d~ LW, CAM               91        91
## 5    192985 https://sof~ K. De Bru~ Kevin De~ CM, CAM               91        91
## 6    200389 https://sof~ J. Oblak   Jan Oblak GK                    91        93
## # i 103 more variables: value_eur <dbl>, wage_eur <dbl>, age <dbl>, dob <date>,
## #   height_cm <dbl>, weight_kg <dbl>, club_team_id <dbl>, club_name <chr>,
## #   league_name <chr>, league_level <dbl>, club_position <chr>,
## #   club_jersey_number <dbl>, club_loaned_from <chr>, club_joined <date>,
## #   club_contract_valid_until <dbl>, nationality_id <dbl>,
## #   nationality_name <chr>, nation_team_id <dbl>, nation_position <chr>,
## #   nation_jersey_number <dbl>, preferred_foot <chr>, weak_foot <dbl>, ...
```

We can see that the data is huge and will require some cleaning before we can start building models with it.

Position Data Processing

We note that players can play in various positions. This could cause problems in terms of classification. We thus remove players that play in more than 1 class, that is defence and midfield or midfield and forward

```r
# Extracting unique positions
positions <- strsplit(as.character(data$player_positions), split = ",")
flattened_positions <- unlist(positions)
trimmed_positions <- trimws(flattened_positions)
unique_positions <- unique(trimmed_positions)

print(unique_positions)
```

```
##  [1] "RW"  "ST"  "CF"  "LW"  "CAM" "CM"  "GK"  "CDM" "LM"  "CB"  "RB"  "RM"
## [13] "LB"  "RWB" "LWB"
```

```r
keeper <- c("GK")
midfield <- c("CAM", "CM", "CDM", "LM", "RM")
defender <- c("CB", "RB", "LB", "RWB", "LWB")
forward <- c("RW", "LW", "ST", "CF")

# Categorize each player's positions
data$category_keeper <- sapply(strsplit(as.character(data$player_positions), ", "), function(pos) any(po
data$category_midfield <- sapply(strsplit(as.character(data$player_positions), ", "), function(pos) any
data$category_defender <- sapply(strsplit(as.character(data$player_positions), ", "), function(pos) any
data$category_forward <- sapply(strsplit(as.character(data$player_positions), ", "), function(pos) any(p

data$num_categories <- rowSums(data[, c("category_keeper", "category_midfield", "category_defender", "ca

# Filter players who belong to only one category
subset_data <- data[data$num_categories == 1, ]
```

Each player is now categorized into one of the four main positions:

Goalkeepers (GK) Defenders (D) Midfielders (M) Forwards (F)

This categorization will serve as the labels for our supervised learning. However, we need to ensure the cleanliness of our data before delving into the modeling. Some of the features are not important in prediction such as name and can be left out in further processing.

We also need to ensure that there is no missing data before we can proceed. For example, only goalkeepers have goalkeeping speed. We can set that to 0 for the other positions for simplicity.

```
classification_columns <- c(
       "overall", "potential",
        "age", "height_cm",
       "weight_kg", "nationality_id", "preferred_foot",
       "weak_foot", "skill_moves", "international_reputation", "work_rate",
       "body_type", "player_tags",
       "player_traits", "pace", "shooting", "passing",
       "dribbling", "defending", "physic", "attacking_crossing",
       "attacking_finishing", "attacking_heading_accuracy", "attacking_short_passing", "attacking_volle
       "skill_dribbling", "skill_curve", "skill_fk_accuracy", "skill_long_passing",
       "skill_ball_control", "movement_acceleration", "movement_sprint_speed", "movement_agility",
       "movement_reactions", "movement_balance", "power_shot_power", "power_jumping",
       "power_stamina", "power_strength", "power_long_shots", "mentality_aggression",
       "mentality_interceptions", "mentality_positioning", "mentality_vision", "mentality_penalties",
       "mentality_composure", "defending_marking_awareness", "defending_standing_tackle", "defending_sl
       "goalkeeping_diving", "goalkeeping_handling", "goalkeeping_kicking", "goalkeeping_positioning",
       "goalkeeping_reflexes", "goalkeeping_speed", "ls", "st",
       "rs", "lw", "lf", "cf",
       "rf", "rw", "lam", "cam",
       "ram", "lm", "lcm", "cm",
     "rcm", "rm", "lwb", "ldm",
       "cdm", "rdm", "rwb", "lb",
       "lcb", "cb", "rcb", "rb",
      "gk", "category_keeper", "category_midfield",
     "category_defender", "category_forward", "num_categories"
)


subset_data <- subset_data[, classification_columns]

subset_data$goalkeeping_speed[is.na(subset_data$goalkeeping_speed)] <- 0

subset_data$pace[is.na(subset_data$pace)] <- 0

subset_data$shooting[is.na(subset_data$shooting)] <- 0
subset_data$passing[is.na(subset_data$passing)] <- 0
subset_data$dribbling[is.na(subset_data$dribbling)] <- 0
subset_data$defending[is.na(subset_data$defending)] <- 0
subset_data$physic[is.na(subset_data$physic)] <- 0

subset_data$player_tags[is.na(subset_data$player_tags)] <- ""
subset_data$player_traits[is.na(subset_data$player_traits)] <- ""

null_counts <- sapply(subset_data, function(x) sum(is.na(x)))
null_counts
```

```
##                     overall                   potential
##                           0                           0
##                         age                   height_cm
##                           0                           0
##                   weight_kg              nationality_id
##                           0                           0
##              preferred_foot                   weak_foot
##                           0                          0
##                 skill_moves   international_reputation
##                           0                          0
##                   work_rate                  body_type
##                           0                          0
##                 player_tags              player_traits
##                           0                          0
##                        pace                   shooting
##                           0                          0
##                     passing                  dribbling
##                           0                          0
##                   defending                     physic
##                           0                          0
##          attacking_crossing        attacking_finishing
##                           0                          0
##  attacking_heading_accuracy    attacking_short_passing
##                           0                          0
##           attacking_volleys             skill_dribbling
##                           0                          0
##                 skill_curve           skill_fk_accuracy
##                           0                          0
##          skill_long_passing          skill_ball_control
##                           0                          0
##       movement_acceleration      movement_sprint_speed
##                           0                          0
##            movement_agility          movement_reactions
##                           0                          0
##            movement_balance            power_shot_power
##                           0                          0
##               power_jumping               power_stamina
##                           0                          0
##              power_strength            power_long_shots
##                           0                          0
##        mentality_aggression    mentality_interceptions
##                           0                          0
##       mentality_positioning           mentality_vision
##                           0                          0
##         mentality_penalties        mentality_composure
##                           0                          0
## defending_marking_awareness   defending_standing_tackle
##                           0                          0
##    defending_sliding_tackle          goalkeeping_diving
##                           0                          0
##        goalkeeping_handling         goalkeeping_kicking
##                           0                          0
##     goalkeeping_positioning       goalkeeping_reflexes
##                           0                          0
##           goalkeeping_speed                          ls
##                           0                          0
##                          st                          rs
##                           0                          0
##                          lw                          lf
##                           0                          0
##                          cf                          rf
##                        0                        0
##                       rw                      lam
##                        0                        0
##                      cam                      ram
##                        0                        0
##                       lm                      lcm
##                        0                        0
##                       cm                      rcm
##                        0                        0
##                       rm                      lwb
##                        0                        0
##                      ldm                      cdm
##                        0                        0
##                      rdm                      rwb
##                        0                        0
##                       lb                      lcb
##                        0                        0
##                       cb                      rcb
##                        0                        0
##                       rb                       gk
##                        0                        0
##          category_keeper        category_midfield
##                        0                        0
##        category_defender         category_forward
##                        0                        0
##           num_categories
##                        0
```

Two more columns have a wealth of information that we can extract and use for prediction

```
subset_data[, c("player_tags", "player_traits")]
```

```
## # A tibble: 15,336 x 2
##    player_tags                                                      player_traits
##    <chr>                                                            <chr>
##  1 "#Dribbler, #Distance Shooter, #FK Specialist, #Acrobat, #Clin~ Finesse Shot~
##  2 "#Aerial Threat, #Distance Shooter, #Clinical Finisher, #Compl~ Solid Player~
##  3 "#Aerial Threat, #Dribbler, #Distance Shooter, #Crosser, #Acro~ Power Free-K~
##  4 "#Dribbler, #Playmaker, #Engine, #Distance Shooter, #Crosser, ~ Injury Prone~
##  5 ""                                                               GK Long Thro~
##  6 "#Speedster, #Dribbler, #Acrobat, #Clinical Finisher, #Complet~ Flair, Speed~
##  7 ""                                                               Leadership, ~
##  8 ""                                                               Rushes Out O~
##  9 "#Distance Shooter, #Clinical Finisher"                          Leadership, ~
## 10 "#Tackling, #Tactician "                                         Leadership
## # i 15,326 more rows
```

The player tags are specific for certain positions which would further help the model in clustering/classification. For example, if a player is a "Distance Shooter", it can be deduced that the player is a forward.

We can extract the player characteristics as follows:

```
# Split each entry by the comma and unlist
tags_list <- unlist(strsplit(subset_data$player_tags, ", "))

# Extract unique tags
unique_tags <- unique(tags_list)

print(unique_tags)
```

```
##  [1] "#Dribbler"            "#Distance Shooter"    "#FK Specialist"
##  [4] "#Acrobat"             "#Clinical Finisher"   "#Complete Forward"
##  [7] "#Aerial Threat"       "#Crosser"             "#Playmaker"
## [10] "#Engine"              "#Complete Midfielder" "#Speedster"
## [13] "#Tackling"            "#Tactician "          "#Poacher"
## [16] "#Tactician"           "#Strength"            "#Complete Defender"
## [19] "#Tackling "           "#Playmaker "
```

```
# Number of players
num_players <- nrow(subset_data)

# Initialize a matrix of FALSE values
tags_matrix <- matrix(FALSE, nrow = num_players, ncol = length(unique_tags))
colnames(tags_matrix) <- unique_tags

# Fill in the matrix with TRUE where the tag is present for the player
for (i in 1:length(unique_tags)) {
  tag <- unique_tags[i]
  tags_matrix[, i] <- grepl(tag, subset_data$player_tags)
}

# Convert the matrix to a dataframe for easier viewing
tags_df <- as.data.frame(tags_matrix)

# Add the new columns to the original dataframe
subset_data <- cbind(subset_data, tags_df)

# Extract and split the player traits
traits_list <- strsplit(as.character(subset_data$player_traits), ", ")

# Extract unique traits
unique_traits <- unique(unlist(traits_list))

# Create a dataframe for traits
traits_df <- data.frame(matrix(ncol = length(unique_traits), nrow = nrow(subset_data)))
colnames(traits_df) <- unique_traits

# Fill in the dataframe with TRUE or FALSE values
for (i in 1:nrow(subset_data)) {
  for (trait in unique_traits) {
    traits_df[i, trait] <- trait %in% traits_list[[i]]
  }
}
```

```r
# Add the new columns to the original dataframe
subset_data <- cbind(subset_data, traits_df)

subset_data <- subset_data[, !(names(subset_data) %in% c("player_tags", "player_traits"))]
```

Some columns are in text format but they can be better represented in a numerical format.

```r
head(subset_data[, c("ls", "st", "rs", "lam", "cam", "ram", "lm", "lcm", "cm", "rcm", "rm",
                     "lwb", "ldm", "cdm", "rdm", "rwb", "lb", "lcb", "cb", "rcb", "rb", "gk")])
```

```
##     ls   st   rs  lam  cam  ram   lm  lcm   cm  rcm   rm  lwb  ldm  cdm  rdm
## 1 89+3 89+3 89+3   93   93   93 91+2 87+3 87+3 87+3 91+2 66+3 64+3 64+3 64+3
## 2 90+2 90+2 90+2 86+3 86+3 86+3 84+3 80+3 80+3 80+3 84+3 64+3 66+3 66+3 66+3
## 3 90+1 90+1 90+1 86+3 86+3 86+3 86+3 78+3 78+3 78+3 86+3 63+3 59+3 59+3 59+3
## 4 83+3 83+3 83+3 89+2 89+2 89+2 89+2 89+2 89+2 89+2 89+2 79+3 80+3 80+3 80+3
## 5 33+3 33+3 33+3 38+3 38+3 38+3 35+3 38+3 38+3 38+3 35+3 32+3 36+3 36+3 36+3
## 6 89+3 89+3 89+3 89+3 89+3 89+3 89+3 81+3 81+3 81+3 89+3 67+3 63+3 63+3 63+3
##    rwb   lb  lcb   cb  rcb   rb   gk
## 1 66+3 61+3 50+3 50+3 50+3 61+3 19+3
## 2 64+3 61+3 60+3 60+3 60+3 61+3 19+3
## 3 63+3 60+3 53+3 53+3 53+3 60+3 20+3
## 4 79+3 75+3 69+3 69+3 69+3 75+3 21+3
## 5 32+3 32+3 33+3 33+3 33+3 32+3 89+3
## 6 67+3 63+3 54+3 54+3 54+3 63+3 18+3
```

87+3 is equivalent to 88+2 so it just makes better sense to convert those values to numerical as well. There are a few NAs left so we can remove them as well.

```r
cols_to_convert <- c("ls", "st", "rs", "lam", "cam", "ram", "lm", "lcm", "cm", "rcm", "rm",
                     "lwb", "ldm", "cdm", "rdm", "rwb", "lb", "lcb", "cb", "rcb", "rb", "gk")
convert_to_numeric <- function(x) {
  # Split string at "+"
  values <- strsplit(as.character(x), "\\+")
  # Add the two parts together
  sapply(values, function(val) sum(as.numeric(val)))
}
# Apply the function to each column
for (col in cols_to_convert) {
  if (col %in% colnames(subset_data)) {  # Check if column exists in the data
    subset_data[[col]] <- convert_to_numeric(subset_data[[col]])
  }
}

# 1. Identify logical columns
logical_cols <- sapply(subset_data, is.logical)

  # 2. Convert those columns to numerical
subset_data[, logical_cols] <- as.data.frame(lapply(subset_data[, logical_cols], as.numeric))

cols_string_numeric <- c("lw", "lf", "cf", "rf", "rw")
subset_data[, cols_string_numeric] <- lapply(subset_data[, cols_string_numeric], as.numeric)
```

```
cols_to_encode <- c("preferred_foot", "work_rate", "body_type")

subset_data[, cols_to_encode] <- lapply(subset_data[, cols_to_encode], function(col) {
    as.numeric(as.factor(col))
})

# Identify non-numeric columns
non_numeric_cols <- sapply(subset_data, function(col) !is.numeric(col))

# Display the names of non-numeric columns
non_numeric_colnames <- names(subset_data)[non_numeric_cols]
print(non_numeric_colnames)
```

```
## character(0)
```

```
subset_data <- subset_data[!apply(is.na(subset_data), 1, any), ]
```

Now that our data is clean, we can add now label the players.

```
# Initialize the new position column with NA
subset_data$position <- NA

# Assign values to the new position column based on the original category columns
subset_data$position[subset_data$category_keeper == 1] <- "G"
subset_data$position[subset_data$category_defender == 1] <- "D"
subset_data$position[subset_data$category_midfield == 1] <- "M"
subset_data$position[subset_data$category_forward == 1] <- "F"

# Drop the original category columns and num_categories
subset_data <- subset_data[, !(colnames(subset_data) %in% c("num_categories", "category_midfield", "cat

positions <- subset_data$position

subset_data <- subset_data[, !names(subset_data) %in% "position"]
```

Unsupervised Learning

Unsupervised learning refers to a type of machine learning where algorithms are trained on data without labeled responses, aiming to identify underlying patterns or structures within the data. In this context, tools like Principal Component Analysis (PCA), Uniform Manifold Approximation and Projection (UMAP), Non-negative Matrix Factorization (NMF), and t-distributed Stochastic Neighbor Embedding (t-SNE) are frequently employed to reduce dimensionality, visualize high-dimensional data, and discover hidden structures. In our case, so far we have hundred's of features. To test the efficacy of the features, we shall reduce the dimensions to 2 for each of the algorithm for ease of visualisation

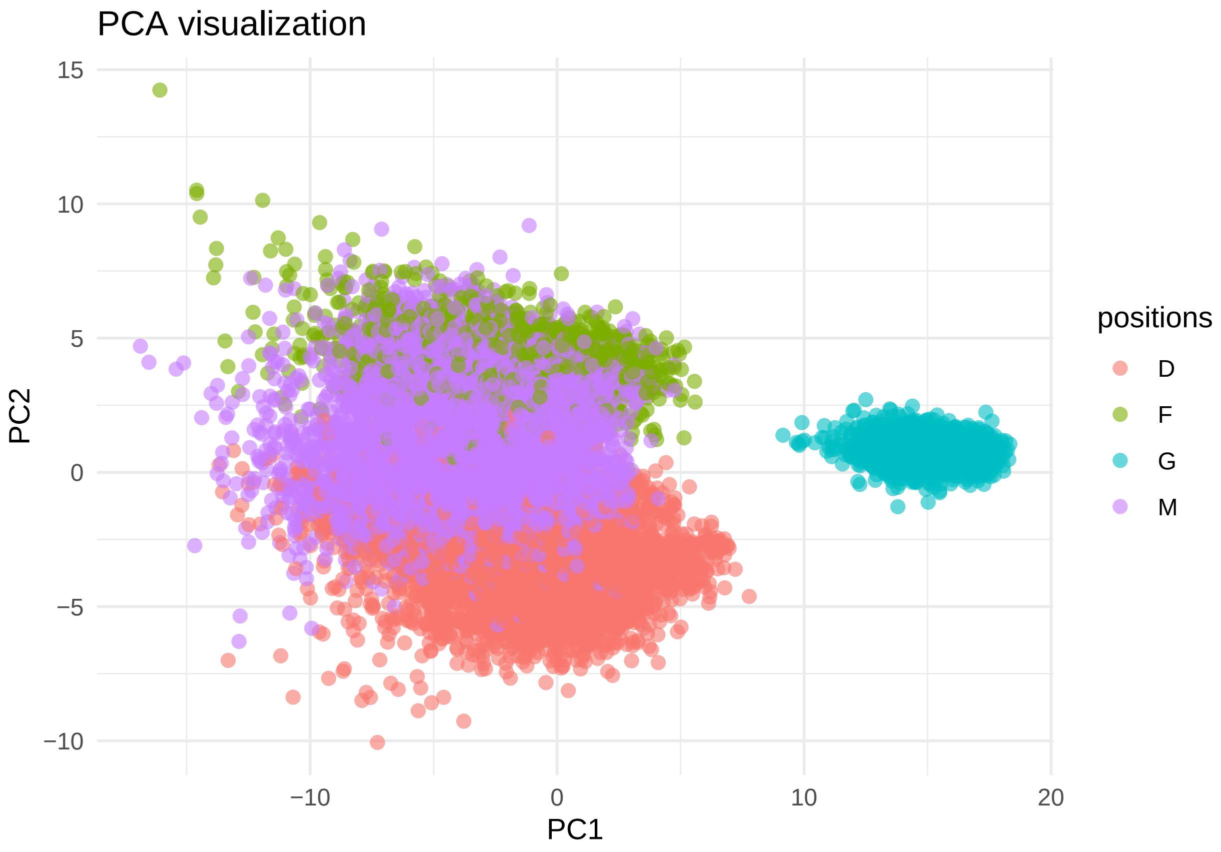
PCA (Principal Component Analysis)

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a dataset while preserving as much variance as possible. By transforming the original data into a new coordinate system, PCA identifies the axes, or principal components, that maximize variance. The first principal component accounts for the most variance, followed by the second, and so on. This process simplifies complex datasets by highlighting their most important features, often facilitating easier visualization and analysis.

```
pca_result <- prcomp(subset_data, center = TRUE, scale. = TRUE)

typeof(pca_result$x)
```

```
## [1] "double"
```

```
df_pca <- as.data.frame(pca_result$x[, 1:2])
df_pca$positions <- positions

ggplot(df_pca, aes(x = PC1, y = PC2, color = positions)) +
  geom_point(alpha = 0.6, size = 2) +
  labs(title = "PCA visualization") +
  theme_minimal()
```

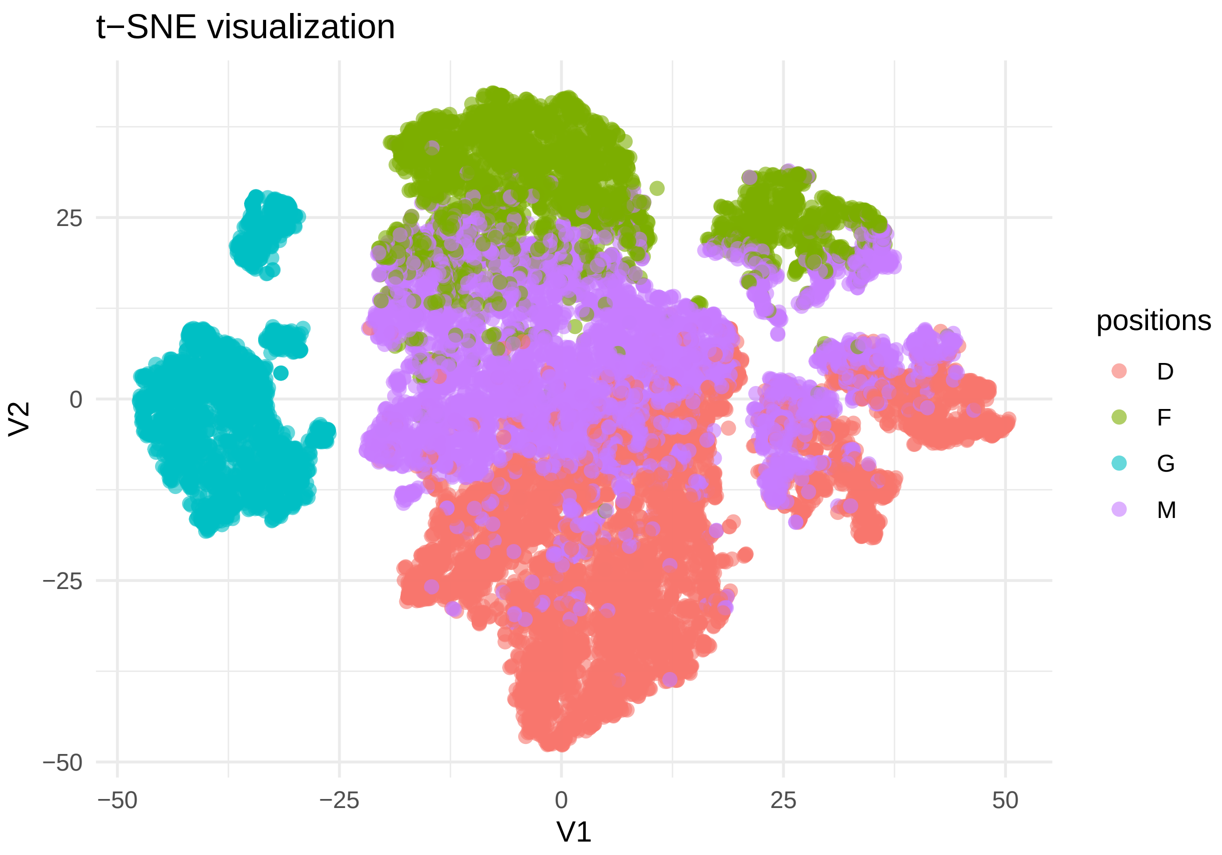


In the PCA visualization, goalkeepers distinctly separate themselves, affirming their unique playing characteristics compared to outfield players. The defenders and forwards, however, are not easily distinguishable, clustering on the sides. This might indicate that, based on the chosen features, their gameplay patterns have similarities. Midfielders, being versatile players often bridging defense and attack, land in the middle and seem to be superimposed upon the defenders and forwards. This “overlay” suggests that midfielders might exhibit characteristics from both groups, but with a prominence that’s distinct to them. The minimal mixing between forwards and defenders indicates some discernible differences between them.

t-SNE (t-Distributed Stochastic Neighbor Embedding):

t-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique especially well-suited for visualizing high-dimensional datasets in two or three dimensions. By minimizing the divergence between two probability distributions—one representing pairwise similarities in the high-dimensional space and the other in the low-dimensional space—t-SNE constructs a map where similar objects are modeled by nearby points, ensuring that clusters of related data points emerge prominently. This makes it particularly effective for data visualization tasks where discerning clusters or groupings is essential.

```r
subset_data <- subset_data[, !names(subset_data) %in% "positions"]

tsne_results <- Rtsne(subset_data, perplexity=30, dims=2)

df_tsne <- as.data.frame(tsne_results$Y)

df_tsne$positions <- df_pca$positions

ggplot(df_tsne, aes(x = V1, y = V2, color = positions)) +
  geom_point(alpha = 0.6, size = 2) +
  labs(title = "t-SNE visualization") +
  theme_minimal()
```

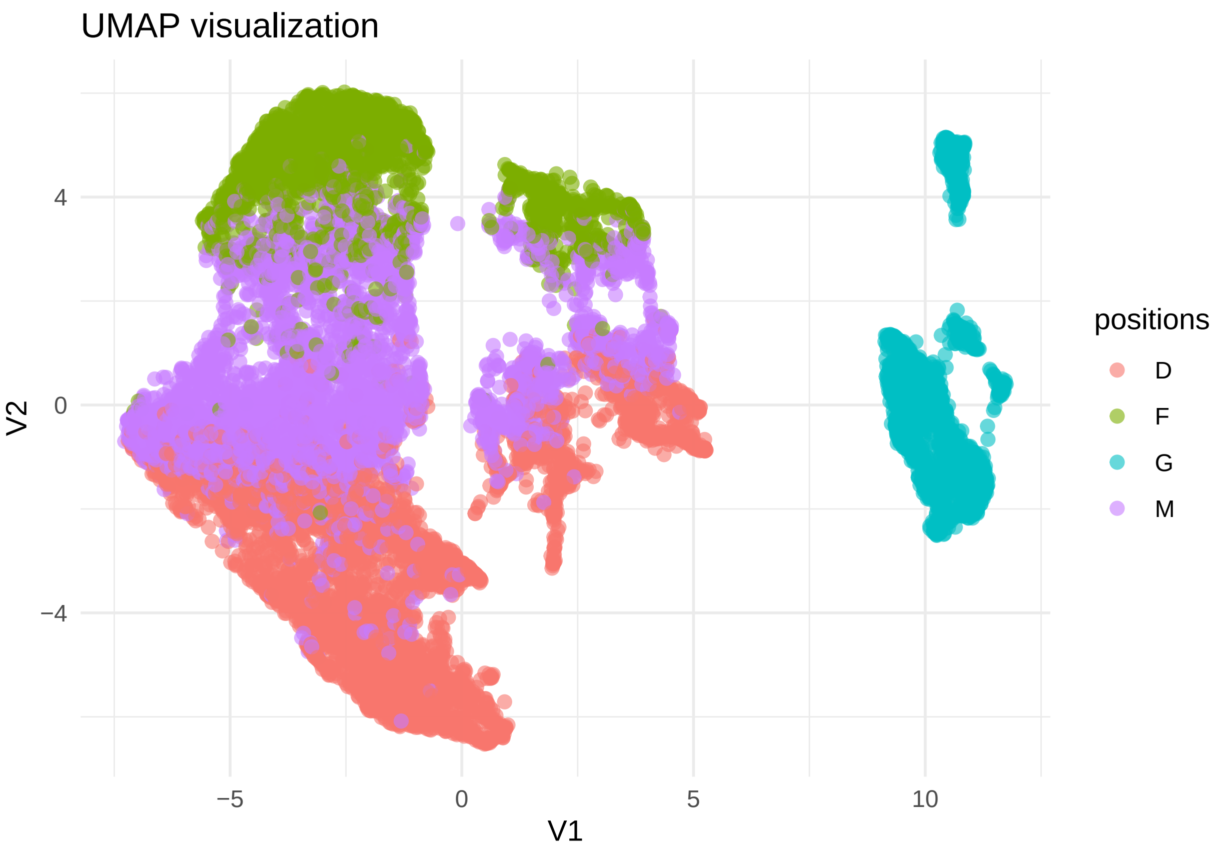


t-SNE has done an impressive job in clustering the goalkeepers, setting them aside in their own distinct clusters, reflecting the specialized nature of their role. The mild intermingling between forwards and midfielders, as well as between midfielders and defenders, suggests overlapping characteristics or skills between these groups. Yet, the fact that there's limited mixing implies that, while there are commonalities, the majority of players within these groups maintain distinct roles on the field.

UMAP (Uniform Manifold Approximation and Projection):

Uniform Manifold Approximation and Projection (UMAP) is a modern dimensionality reduction technique that is especially effective for visualizing high-dimensional data in lower-dimensional spaces, often two or three dimensions. Building on topological data analysis and manifold learning principles, UMAP emphasizes the preservation of both local and global structures in the data. Not only is UMAP faster than techniques like t-SNE for large datasets, but it also offers flexibility with its mathematical foundations, allowing it to be used in tasks beyond mere visualization, such as semi-supervised learning and clustering.

```r
subset_data <- subset_data[, !names(subset_data) %in% "positions"]

umap_results <- umap(subset_data, n_neighbors = 15, min_dist = 0.1, n_components = 2)
df_umap <- as.data.frame(umap_results)

df_umap$positions <- df_pca$positions

ggplot(df_umap, aes(x = V1, y = V2, color = positions)) +
  geom_point(alpha = 0.6, size = 2) +
  labs(title = "UMAP visualization") +
  theme_minimal()
```

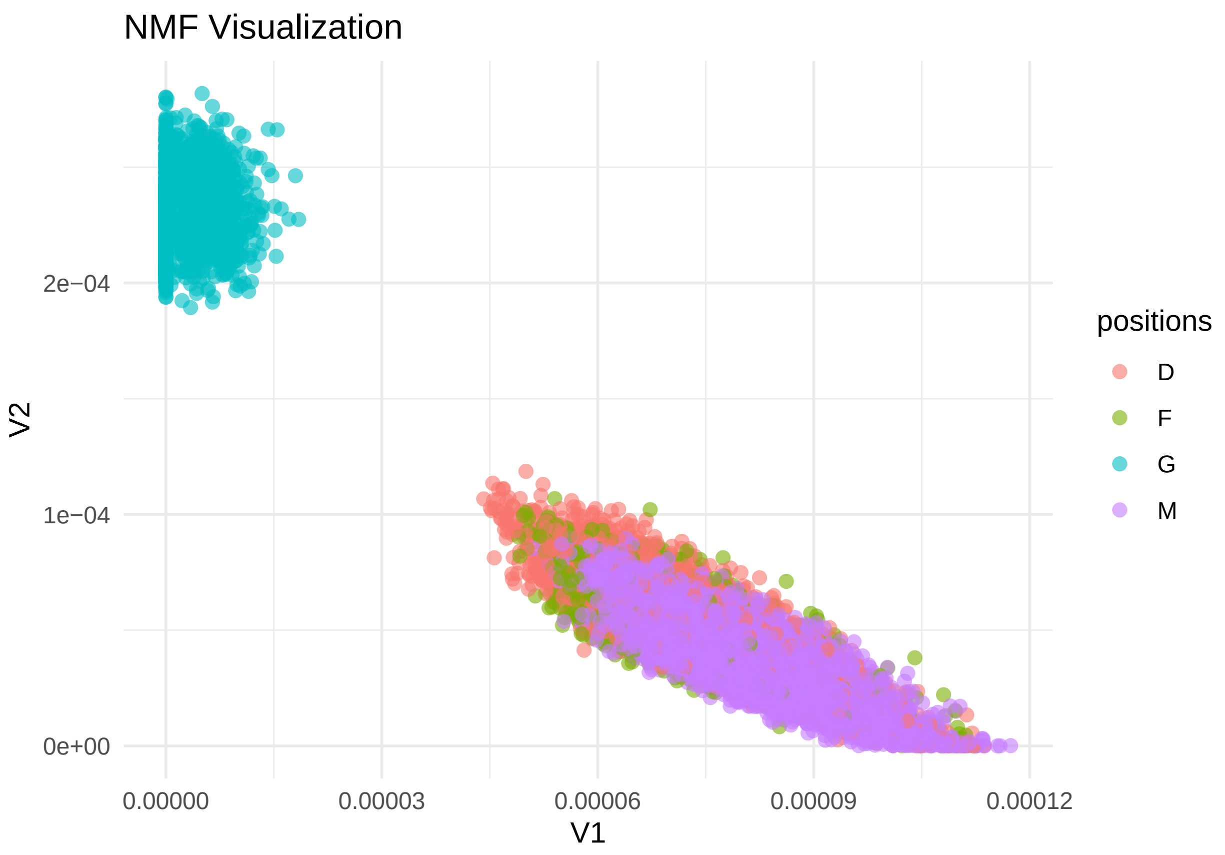


UMAP's quicker computation to t-sne is a significant advantage. Like t-SNE, UMAP sets goalkeepers at a respectable distance, underlining their distinct position in the game. The mix between midfielders and forwards and between midfielders and defenders is slightly less compared to t-SNE. This suggests that UMAP may capture subtle differences between these groups a bit more distinctly, pointing to its potential greater sensitivity or its ability to highlight nuanced differences in the data.

NMF (Non-negative Matrix Factorization):

Non-negative Matrix Factorization (NMF) is a linear dimensionality reduction technique that decomposes a non-negative matrix into two lower-dimensional non-negative matrices, capturing the latent structure in the data. Unlike other methods like PCA, NMF enforces non-negativity constraints, resulting in additive, parts-based representations. This property makes NMF particularly valuable in applications where the components need to be interpretable, such as topic modeling in text analysis or spectral data decomposition, where the factors can represent underlying patterns or topics in the data.

```r
# Ensure all values are non-negative
if(any(subset_data < 0)){
  stop("Data contains negative values. NMF requires non-negative data.")
}

subset_data <- subset_data[, !names(subset_data) %in% "positions"]

nmf_result <- nmf(subset_data, rank = 2, method = "lee", seed = 12345)
W <- basis(nmf_result)
H <- coef(nmf_result)

# Depending on your needs, W (basis matrix) or H (coefficient matrix) can be used for visualization.
# Here, we'll use W for our visualization.
df_nmf <- as.data.frame(W)

df_nmf$positions <- df_pca$positions

ggplot(df_nmf, aes(x = V1, y = V2, color = positions)) +
  geom_point(alpha = 0.6, size = 2) +
  labs(title = "NMF Visualization") +
  theme_minimal()
```



NMF presents a more abstracted view. With goalkeepers clustering together, their distinct role is evident. However, it aggregates midfielders, forwards, and defenders into a singular cluster, hinting that their features might not be adequately differentiated in the representation NMF provides. Despite the closer clustering, the clear distance between goalkeepers and the combined MFD group is noteworthy, emphasizing again the unique characteristics of goalkeepers.

Supervised Learning

Supervised learning is a category of machine learning where algorithms are trained on a labeled dataset, using input-output pairs to learn a mapping from inputs to outputs. In this context, the aim is to find a model that predicts the output values with high accuracy for unseen data. Techniques like Support Vector Machines (SVM) and Random Forest are commonly employed for their ability to handle complex data and produce robust predictions. To evaluate the performance of these models, it's crucial to inspect the confusion matrices, as they provide insights into the types of errors made. By doing so, we can fine-tune our models and better understand the results, ensuring that our predictions are not only accurate but also meaningful in the real-world context.

Support Vector Machine

Support Vector Machines (SVM) are a class of supervised learning algorithms designed for classification and regression tasks. At their core, SVMs aim to find the optimal hyperplane that best separates the data into different classes, especially in cases where the classes are not linearly separable. Using what's known as the "kernel trick," SVMs can transform the input data into a higher-dimensional space, making it possible to find a separating hyperplane even for complex, non-linear datasets. The chosen hyperplane maximizes the margin between the two classes, ensuring that the model has the best generalization capability for unseen data. SVMs are widely celebrated for their robustness, versatility, and ability to handle high-dimensional data effectively.

```
subset_data$positions <- df_pca$positions

# Splitting the data into training and testing sets (70% training, 30% testing)
set.seed(123) # For reproducibility
trainIndex <- createDataPartition(subset_data$positions, p = .7, list = FALSE)
train_data <- subset_data[trainIndex,]
test_data <- subset_data[-trainIndex,]

# 2. Model Selection
model <- train(positions ~ ., data = train_data, method = "svmLinear", trControl = trainControl(method
```

```
## Warning in .local(x, ...): Variable(s) ‘’ constant. Cannot scale data.
```

```
# 3. Training is done in the above step with the train function.
# 4. Testing and Evaluation
predictions <- predict(model, newdata = test_data)
accuracy <- sum(predictions == test_data$position) / nrow(test_data)

predictions <- as.factor(predictions)
test_data$position <- as.factor(test_data$position)

# Compute the confusion matrix for SVM
confusion_matrix_svm <- confusionMatrix(predictions, test_data$position)

# Print the confusion matrix
print(confusion_matrix_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    D    F    G    M
##          D 1507    0    0   46
##          F    1  709    0   61
##          G    0    0  639    0
##          M   43   81    0 1332
##
## Overall Statistics
##
##                Accuracy : 0.9475
##                  95% CI : (0.9405, 0.9539)
##     No Information Rate : 0.351
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9268
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: D Class: F Class: G Class: M
## Sensitivity            0.9716   0.8975   1.0000   0.9256
## Specificity            0.9840   0.9829   1.0000   0.9584
## Pos Pred Value         0.9704   0.9196   1.0000   0.9148
## Neg Pred Value         0.9846   0.9778   1.0000   0.9639
## Prevalence             0.3510   0.1788   0.1446   0.3256
## Detection Rate         0.3410   0.1604   0.1446   0.3014
## Detection Prevalence   0.3514   0.1745   0.1446   0.3295
## Balanced Accuracy      0.9778   0.9402   1.0000   0.9420
```

SVM Results Analysis The confusion matrix presents the results of the SVM classifier for the different soccer player roles: Defenders (D), Forwards (F), Goalkeepers (G), and Midfielders (M).

Defenders (D): The SVM classifier correctly predicted 1507 defenders and misclassified 46 as midfielders. With a sensitivity of 97.16%, the model was highly accurate in identifying defenders, and its positive predictive value of 97.04% indicates a high proportion of true positive predictions for defenders.

Forwards (F): 709 forwards were accurately predicted, but 61 were mistaken as midfielders, and 1 as a defender. The sensitivity for forwards is 89.75%, indicating that the model captured a majority of the actual forwards, and a positive predictive value of 91.96% suggests a solid performance in its forward predictions.

Goalkeepers (G): The classifier exhibited perfect accuracy for goalkeepers with 639 correct predictions and zero misclassifications, leading to a sensitivity and positive predictive value of 100%.

Midfielders (M): For midfielders, 1332 were correctly classified, while 43 and 81 were incorrectly predicted as defenders and forwards, respectively. The sensitivity here is 92.56%, indicating a strong performance, but there seems to be some confusion, especially with forwards.

Overall Performance: The SVM classifier achieved an overall accuracy of 94.75%, indicating that it correctly predicted the player roles in about 95 out of 100 cases. The Kappa statistic, which measures agreement between the predicted and actual classes, is 0.9268, signifying an excellent agreement beyond chance. The balanced accuracy values for each class are close to or at 100%, showcasing the model's robustness across different player roles. The statistics provided, including specificity, negative predictive value, and detection rates, further emphasize the classifier's efficacy.

Random Forests

Random Forests are an ensemble learning method primarily used for classification and regression tasks. They operate by constructing a multitude of decision trees during training and outputting either the mode of the classes (for classification) or the mean prediction (for regression) of individual trees for unseen data. Key to the random forest's success is its ability to introduce randomness into the model building process, which combats overfitting. This is achieved by not only training each tree on a random bootstrap sample of the dataset but also by selecting a random subset of features at each split in the decision tree building process. As a result, Random Forests tend to be more accurate and robust than individual decision trees. They also provide a measure of feature importance, giving insight into which variables are most influential in prediction.

```r
# Correcting column names if they contain problematic characters
colnames(subset_data) <- make.names(colnames(subset_data), unique = TRUE)

subset_data$positions <- df_pca$positions
# Splitting data and training model
set.seed(123) # For reproducibility
trainIndex <- createDataPartition(subset_data$positions, p = .7, list = FALSE)
train_data <- subset_data[trainIndex,]
test_data <- subset_data[-trainIndex,]
```

```
# Convert positions to a factor, if not already
train_data$positions <- as.factor(train_data$positions)
test_data$positions <- as.factor(test_data$positions)

# Train the Random Forest model
rf_model <- randomForest(positions ~ ., data = train_data, ntree = 100)

# Make predictions
rf_predictions <- predict(rf_model, newdata = test_data)

# Compute accuracy
rf_accuracy <- sum(rf_predictions == test_data$positions) / nrow(test_data)
print(paste("Random Forest Accuracy:", round(rf_accuracy * 100, 2), "%"))
```

```
## [1] "Random Forest Accuracy: 93.78 %"
```

```
# Compute the confusion matrix for Random Forest
confusion_matrix_rf <- confusionMatrix(rf_predictions, test_data$positions)

# Print the confusion matrix
print(confusion_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    D    F    G    M
##          D 1499    0    0   72
##          F    0  705    0   66
##          G    0    0  639    0
##          M   52   85    0 1301
##
## Overall Statistics
##
##                Accuracy : 0.9378
##                  95% CI : (0.9302, 0.9447)
##     No Information Rate : 0.351
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9132
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: D Class: F Class: G Class: M
## Sensitivity            0.9665   0.8924   1.0000   0.9041
## Specificity            0.9749   0.9818   1.0000   0.9540
## Pos Pred Value         0.9542   0.9144   1.0000   0.9047
## Neg Pred Value         0.9817   0.9767   1.0000   0.9537
## Prevalence             0.3510   0.1788   0.1446   0.3256
## Detection Rate         0.3392   0.1595   0.1446   0.2944
## Detection Prevalence   0.3555   0.1745   0.1446   0.3254
## Balanced Accuracy      0.9707   0.9371   1.0000   0.9291
```

Random Forest Results Analysis: The presented results showcase the performance of the Random Forest classifier in discerning soccer player roles: Defenders (D), Forwards (F), Goalkeepers (G), and Midfielders (M).

Defenders (D): Out of all predicted defenders, 1499 were correctly identified, while 72 were misclassified as midfielders. The sensitivity of 96.65% suggests that the Random Forest model did a commendable job in picking out defenders, while its positive predictive value of 95.42% indicates a significant proportion of true positive predictions.

Forwards (F): For this group, 705 were predicted correctly, but 66 were mistakenly identified as midfielders. With a sensitivity of 89.24%, the model successfully captured most of the forwards. The positive predictive value of 91.44% is slightly below that of SVM but still shows a strong prediction rate for forwards.

Goalkeepers (G): The classifier achieved perfection in distinguishing goalkeepers with 639 correct predictions and no misclassifications. Both sensitivity and positive predictive value stand at 100%, reiterating the distinct characteristics of goalkeepers compared to outfield players.

Midfielders (M): 1301 midfielders were rightly classified, but there were some confusions, with 52 and 85 of them being misclassified as defenders and forwards, respectively. The sensitivity of 90.41% denotes a solid performance, but there's a noticeable confusion with other positions, especially with forwards.

Overall Performance: Random Forest achieves an overall accuracy of 93.78%, meaning it predicts player roles accurately in approximately 94 out of 100 instances. The Kappa statistic of 0.9132 suggests excellent agreement between predicted and actual classifications, going beyond random chance. Balanced accuracy for each group is near or at 100%, emphasizing the model's consistency across distinct player roles. While Random Forest provides a strong classification for player roles, there's a slight drop in accuracy for outfield players, especially midfielders, compared to the SVM results.

Conclusion

In this tutorial, we embarked on an extensive exploration of both supervised and unsupervised learning techniques to decipher patterns within player data. Our unsupervised analysis, spanning PCA, t-SNE, UMAP, and NMF, unearthed intriguing clusters and relationships among player roles. Goalkeepers consistently emerged as a distinct category across methodologies, highlighting the specialized skills inherent to their role. Meanwhile, the overlap observed among outfield players, especially between midfielders and other positions, reflects the multifaceted nature of their gameplay.

Switching gears to the supervised realm, SVM and Random Forest algorithms were employed. Both classifiers showcased robust accuracy rates, with SVM slightly edging out Random Forest in certain classifications. Goalkeepers, once again, were pinpointed with impeccable precision, while outfield players presented subtle challenges in distinguishing their roles, reaffirming findings from the unsupervised methods.

The combined insights gleaned from this study accentuate the dynamic and multi-dimensional qualities of soccer player data. By juxtaposing supervised and unsupervised paradigms, we gained a comprehensive understanding of player attributes, revealing both distinct demarcations and intertwined connections in their in-game roles. This research not only deepens our appreciation of the intricacies within the FIFA 23 player dataset but also underscores the potential of melding diverse machine learning approaches to illuminate complex datasets in sports analytics and beyond.