Report of Knowledge Discovery and Data Mining'

Data Mining

1/ Problem Understanding

The dataset I used was created by Stefano Leone and is composed of seven files. Indeed, each file represent a version of the game fifa by year from 15 to 20 and includes all the players data for the Career Mode. The one that interest us and that I choose is the fifa 20 file because it's the most recent one. So, we end up with 104 variables for 18278 observations. The dataset has been done by scraping the website https://sofifa.com.

The main problem I wanted to tackle was a classification problem. From all the attributes of a player is it possible to find his position in a team. In order to do it I need to gather as much attributes and data to train a model on and do a classification task.

2/ Data Understanding and Preparation

The data needed to be clean and prepared in order to do my classification. Indeed, the first thing I did is selecting all the variables that will be useful for my classification. All the code can be found and execute in the file code.R

At that step I made the choice to don't manage the goalkeeper because the way they are implemented in fifa needed me to take them apart, either to normalize their missing value (a lot missing) or adding them some them don't have like the casual field's players.

This was done by selecting all the columns of skill that I had and selecting all payer except goalkeepers. Then the other problem I had to solve was to change the datatype so I could exploit them later. All of this work was done by this function:

```
players_20 <- players_20 %>%
  select(selected) %>%
  filter(!is.na(club), !is.na(team_position), team_position != "SUB",team_position != "GK")

players_20$team_position <- gsub("[0-9]","",players_20$team_position)

for (val in c(2,8,10)){
  players_20[[val]] <- as.character(players_20[[val]])
  players_20[[val]] <- as.numeric(players_20[[val]])

}
```

Then I had the idea to add an attribute that could be easily compute thanks to the data's that we had on a player. This was the Body Mass Index (or BMI) that is define by: **BMI** = weight (kg) / [height (m)]^2

Finally, the last step was to create more general class to do my position classification job faster and easier. I decided to regroup all the soccer position in 3 groups:

1: Defender	2: Middle fielder	3: Attacker
-Left Back -Center Back -Right Back -Defense Mid	-Left Wing -Right Wing -Midfield	-Attack Mid -Forward

This job was done by the function mutate and here is the extract of the code:

```
players_20 <- players_20 %>%
    mutate(Class = case_when(PositionNames == "Left Back" ~ '1',
                             PositionNames == "Center Back" ~ '1',
                             PositionNames == "Right Back" ~ '1',
                             PositionNames == "Attack Mid" ~ '3',
                             PositionNames == "Left Wing" ~ '2',
                             PositionNames == "Forward" ~ '3',
                             PositionNames == "Right Wing" ~ '2',
                             PositionNames == "Defense Mid" ~ '1',
                             PositionNames == "Midfield" ~ '2'))
```

Finally, I decided to do a work on my attributes to make a selection on the best of them. This selection was really important for the next job of modelling, in order to avoid collinearity, speed up the computing time. To get an idea of the links between my variables I did the correlation matrix and heatmap: (pictures are available in bigger size at the end of the document or in the picture folder)

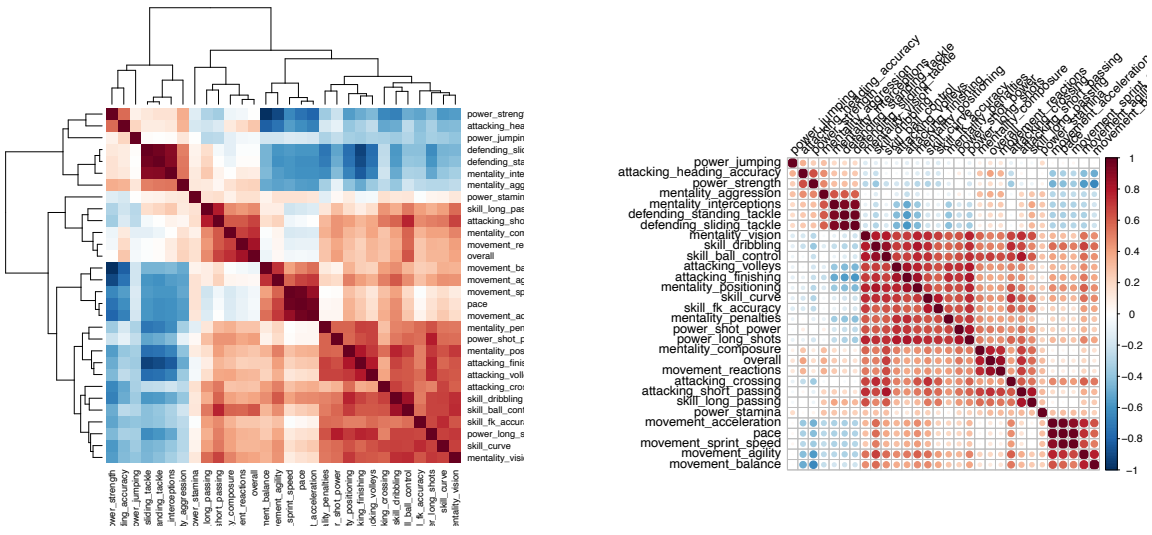
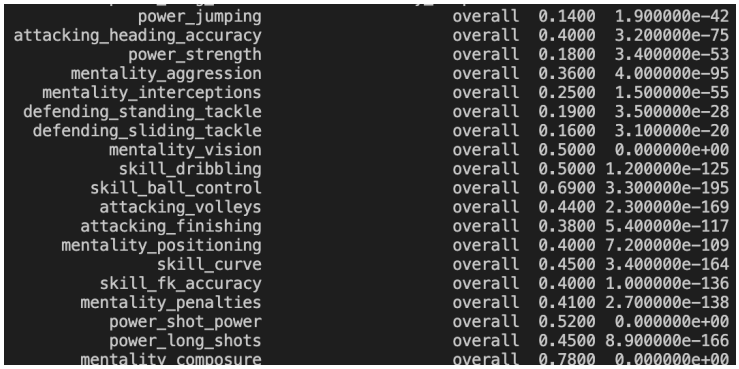


Figure 1: Heatmap and correlation matrix between the variables

As we can see in the graph below there are plenty of properties that correlate. To ma ke my selection I choose the one that were the most correlated to the overall of a pla yer. Indeed, the overall of a player is usually the first thing we see and check when lo oking to a player on FIFA. Here are the variables correlated to overall:



power_jumping	overall	0.1400	1.900000e-42
attacking_heading_accuracy	overall	0.4000	3.200000e-75
power_strength	overall	0.1800	3.400000e-53
mentality_aggression	overall	0.3600	4.000000e-95
mentality_interceptions	overall	0.2500	1.500000e-55
defending_standing_tackle	overall	0.1900	3.500000e-28
defending_sliding_tackle	overall	0.1600	3.100000e-20
mentality_vision	overall	0.5000	0.000000e+00
skill_dribbling	overall	0.5000	1.200000e-125
skill_ball_control	overall	0.6900	3.300000e-195
attacking_volleys	overall	0.4400	2.300000e-169
attacking_finishing	overall	0.3800	5.400000e-117
mentality_positioning	overall	0.4000	7.200000e-109
skill_curve	overall	0.4500	3.400000e-164
skill_fk_accuracy	overall	0.4000	1.000000e-136
mentality_penalties	overall	0.4100	2.700000e-138
power_shot_power	overall	0.5200	0.000000e+00
power_long_shots	overall	0.4500	8.900000e-166
mentality_composure	overall	0.7800	0.000000e+00

So this lead me to my unexpected and interesting choice of variables and final data set clean and ready to use for my models : model_data composed of : mentality_com posure, skill_ball_control, power_shot_power, mentality_vision, skill_dribbling, skill_c urve, Class. This also surprise me to see that in general for a player his overall is mai nly due to his mentality_composture, I would have expected strength, speed or stami na in the first position.

4/ Modeling, Evaluation and Deployment

I did two approaches to tackle my classification problem, first one with the Knn algorithm and the second with support vector machines. All of these methods were implemented and deployed with the help of caret library which is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for: data splitting, pre-processing, feature selection, model tuning using resampling and variable importance estimation.

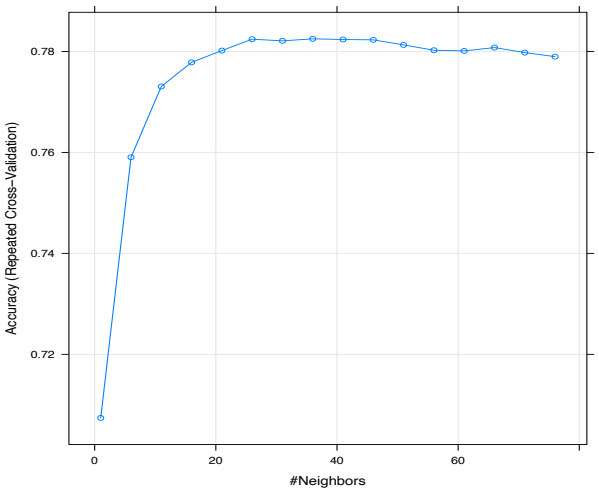
In my case I took an 80/20 split with the createDataPartition() that also try to balance the class distribution so that they are similar in both the training and test set. Then the first important thing to do was to define some basic control parameters for the training step with the function trainControl. The arguments are : “*method” that* defines the resampling method during training, “*number”* sets the “k” in k-fold and ”*repeats”* defines how often the cross validation should be repeated.

I used standard value in found on the internet and my courses for repears=”repeatdcv” to do the k-fold cross validation, number = 10 and repeat = 3. The next function is train() function is used to train our model, his arguments is *preProcess* that is used to scale and center the data, and tuneGrid() defines a grid of hyperparameters to be used in the training.

4.1/ Knn

Knn is a very strong algorithm where the only hyperparameter of this model is k, the number of considered neighbors. So, I did a lot of attempt with different K

values to get the best result. I tried with a tunegrid, and the help of the function seq(), between 1 and 80 with a step of 5. Here are the results I get:




```
k-Nearest Neighbors

5016 samples
  30 predictor
   3 classes: '1', '2', '3'

Pre-processing: centered (30), scaled (30)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 4514, 4514, 4515, 4513, 4516, 4513, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.7074058  0.4920091
   6  0.7555863  0.5615041
  11  0.7690829  0.5807481
  16  0.7738693  0.5875685
  21  0.7774605  0.5923901
  26  0.7797805  0.5957690
  31  0.7805774  0.5966535
  36  0.7808436  0.5965377
  41  0.7787817  0.5923486
  46  0.7806441  0.5955107
  51  0.7795789  0.5932917
  56  0.7789158  0.5919414
  61  0.7792466  0.5923897
  66  0.7783167  0.5903254
  71  0.7781179  0.5898780
  76  0.7776529  0.5888689

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 36.
```

We can see the that the k = 36 is the best result with an accuracy of 0,780 and a Kappa = 0,5965. Kappa statistic is a measurement of the agreement for categorical items.

After the training, The function predict() allow us to see the result. The resulting performance can then be viewed with the function confusionMatrix() :

```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 592  11  97
         2   0   0   1
         3  49 112 390

Overall Statistics

               Accuracy : 0.7843
                 95% CI : (0.7605, 0.8068)
    No Information Rate : 0.512
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6022

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 1  Class: 2 Class: 3
Sensitivity            0.9236 0.0000000   0.7992
Specificity            0.8232 0.9991143   0.7893
Pos Pred Value         0.8457 0.0000000   0.7078
Neg Pred Value         0.9112 0.9016787   0.8602
Prevalence             0.5120 0.0982428   0.3898
Detection Rate         0.4728 0.0000000   0.3115
Detection Prevalence   0.5591 0.0007987   0.4401
Balanced Accuracy      0.8734 0.4995571   0.7942
```

4.1/ SVM

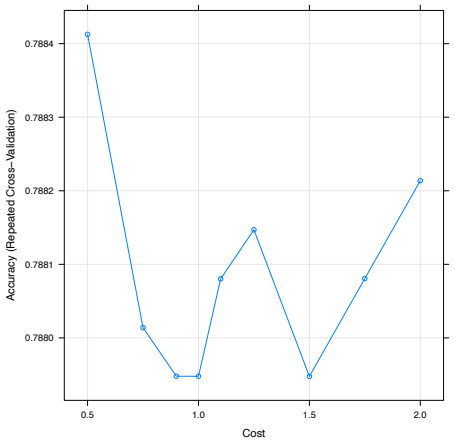
I decided to use the package of caret that include a linear SVM here. The hyperparameter to be set is a cost value, which determines the margin around the

separating hyperplanes (the bigger the value, the smaller the margin). So again we let the same trainControl() function but this time with an linear SVM

```
grid_svm <- expand.grid(.cost=c(0.5, 0.75, 0.9, 1, 1.1, 1.25, 1.5, 1.75, 2))
fifa_svm_linear <- train(Class ~., data = train_data, method = "svmLinear2",
                         trControl=train_control,
                         preProcess = c("center", "scale"),
                         tuneGrid = grid_svm)

fifa_svm_linear

plot(fifa_svm_linear)

fifa_svm_linear_predict <- predict(fifa_svm_linear,newdata = test_data)
confusionMatrix(fifa_svm_linear_predict,test_data$Class)
```

Again, I tuned the cost parameters thanks to examples I saw on internet and from my courses.

After the computing I obtain these results:




```
Support Vector Machines with Linear Kernel

5016 samples
  30 predictor
   3 classes: '1', '2', '3'

Pre-processing: centered (30), scaled (30)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 4515, 4515, 4513, 4514, 4516, 4515, ...
Resampling results across tuning parameters:

  cost  Accuracy   Kappa
  0.50  0.7884126  0.6100659
  0.75  0.7880139  0.6093443
  0.90  0.7879479  0.6092229
  1.00  0.7879478  0.6092229
  1.10  0.7880803  0.6094869
  1.25  0.7881470  0.6096045
  1.50  0.7879477  0.6092312
  1.75  0.7880807  0.6094794
  2.00  0.7882137  0.6097236

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cost = 0.5.
```

We can see the that the cost = 0.5 is the best result with an accuracy of 0,788.

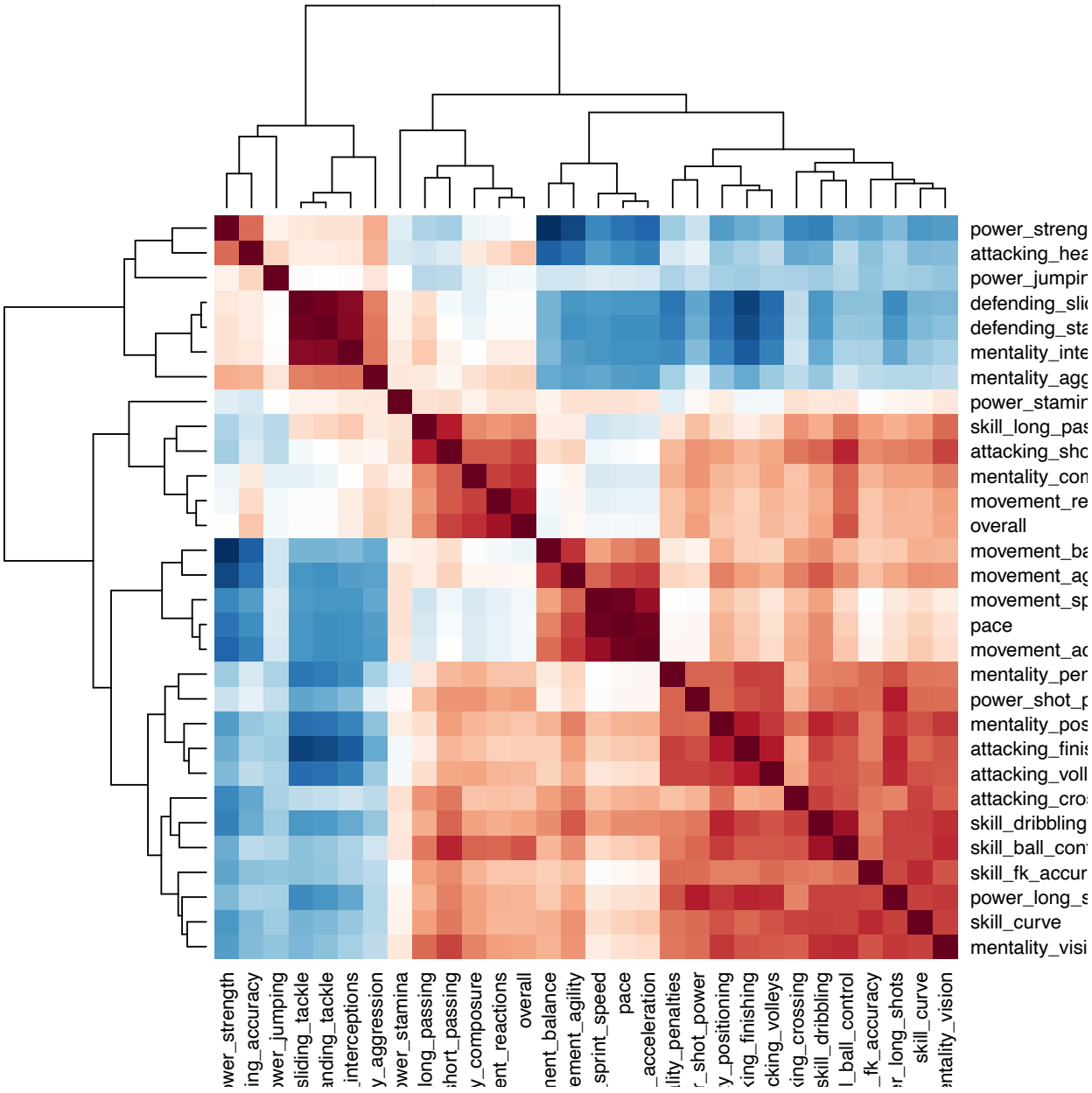
After the training, again function predict() allow us to see the result. The result with the function confusionMatrix() is :

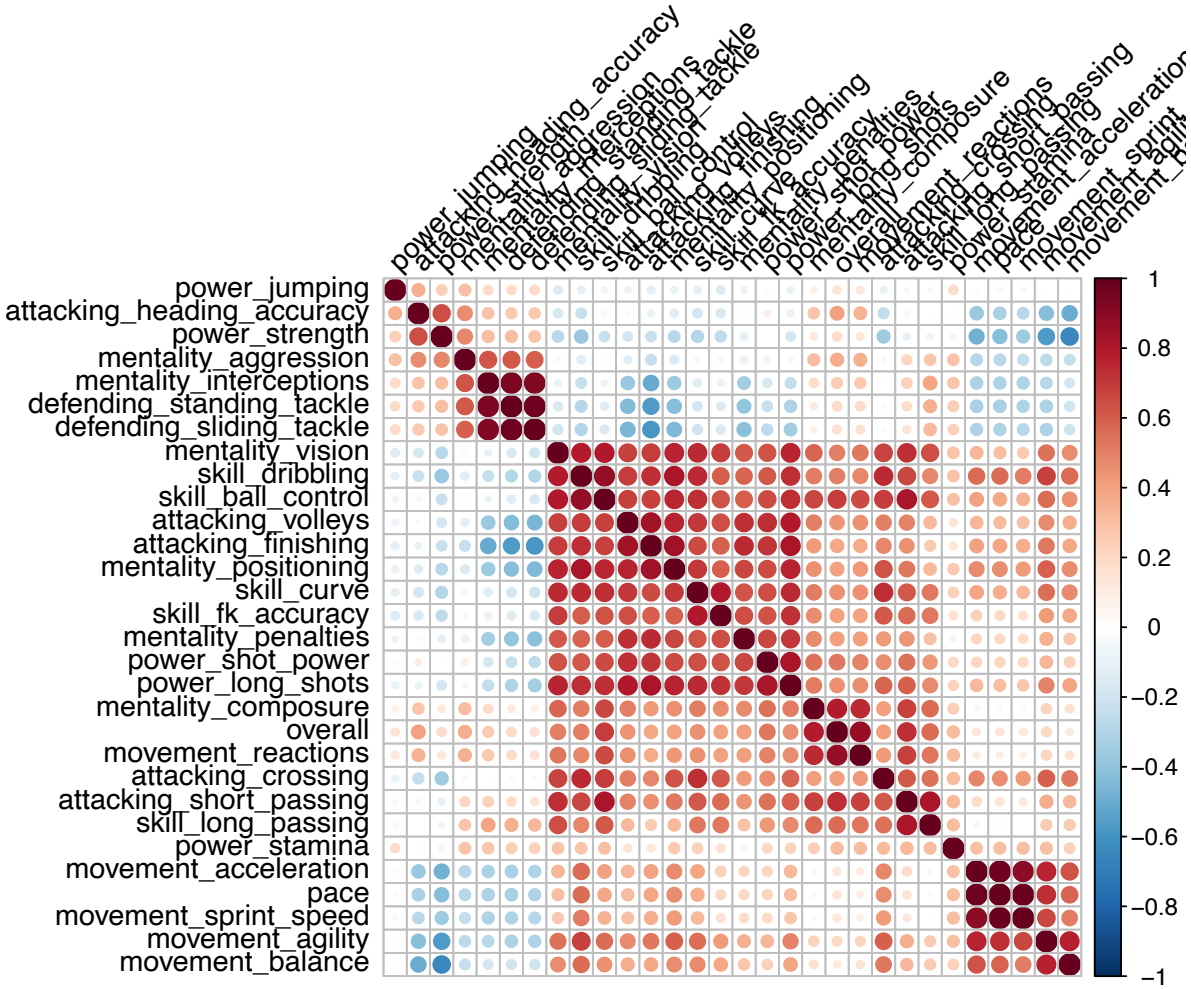
```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 593  10  93
         2   0   0   0
         3  48 113 395

Overall Statistics

               Accuracy : 0.7891
                 95% CI : (0.7655, 0.8114)
    No Information Rate : 0.512
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6112

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.9251  0.00000   0.8094
Specificity            0.8314  1.00000   0.7893
Pos Pred Value         0.8520      NaN   0.7104
Neg Pred Value         0.9137  0.90176   0.8664
Prevalence             0.5120  0.09824   0.3898
Detection Rate         0.4736  0.00000   0.3155
Detection Prevalence   0.5559  0.00000   0.4441
Balanced Accuracy      0.8783  0.50000   0.7993
```

5/ Conclusion

In my result we can see one major problem coming from the class 2, I think it come from my initial clean dataset that contains only a few examples of it and during the split sometimes create errors. I could fix it, by taking a look at my initial data set and see the proportion of each class in it. Otherwise for the other classes we can see that we had some satisfying result for the other class. If it was a competition between our models, we can see that the result of the SVM is a little bit more that the Knn. Finally, to go further, I only use the initial function of the possibility offered by the caret package, many tuning possibilities or hyperparameter could have been explored and tried. I choose this dataset and topic because I love soccer and all the possibility that it offers. It was interesting to see the videogames I played for years from another point of view. I can conclude from my study, that now I know what characteristic I will look for a player before buying him and also that data analysis linked to soccer is a very powerful and strategic advantage that professional club use a lot.

Appendix


power_streng
attacking_hea
power_jumpir
defending_sli
defending_sta
mentality_inte
mentality_agg
power_stamir
skill_long_pas
attacking_sho
mentality_con
movement_re
overall
movement_ba
movement_ag
movement_sp
pace
movement_ac
mentality_per
power_shot_p
mentality_pos
attacking_finis
attacking_voll
attacking_cro
skill_dribbling
skill_ball_con
skill_fk_accur
power_long_s
skill_curve
mentality_visi
ower_strength
ing_accuracy
ower_jumping
sliding_tackle
anding_tackle
_interceptions
y_aggression
ower_stamina
long_passing
hort_passing
y_composure
ent_reactions
overall
nent_balance
ement_agility
sprint_speed
pace
_acceleration
lity_penalties
r_shot_power
y_positioning
king_finishing
cking_volleys
king_crossing
skill_dribbling
l_ball_control
_fk_accuracy
er_long_shots
skill_curve
ntality_vision


power_jumping
attacking_heading_accuracy
power_strength
mentality_aggression
mentality_interceptions
defending_standing_tackle
defending_sliding_tackle
mentality_vision
skill_dribbling
skill_ball_control
attacking_volleys
attacking_finishing
mentality_positioning
skill_curve
skill_fk_accuracy
mentality_penalties
power_shot_power
power_long_shots
mentality_composure
overall
movement_reactions
attacking_crossing
attacking_short_passing
skill_long_passing
power_stamina
movement_acceleration
pace
movement_sprint_speed
movement_agility
movement_balance
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1