Football Manager : Predicting a player's position on the field

Mohamed Amine Hachicha, Khalil Bergaoui, Imen Ayadi, Zeineb Letaief

7th January 2019

1 Abstract

Data science and machine learning are nowadays not only used in predicting sales and building models for business companies but are also exploited in many other fields. If we take the example of football (or soccer if you're American), machine learning is exploited to challenge bookmakers when it comes to predicting match results based, and specialist journalists when it comes to giving a market value for a given player based on its performance or even assign a rating to a player based on its performance. Our project idea comes as a result to a common observation made by both football fans and professional: many young players do not play in the best adequate position for them given their different abilities and characteristics at the very beginning of their careers, which results in many valuable years wasted playing in the wrong position on the football pitch and it's generally very late when they discover this potential. Our project is never aiming to give a complete solution to this issue given its complexity and that we're not football experts but it can be considered as a first small step to doing so, especially when we saw that there are very few people who ever thought of using machine learning algorithms to predict the position of a player given its statistics.

2 Introduction

Football (soccer) is a team sport where each team consists of 11 players and each player occupies a well-known position on the pitch: Goalkeeper (GK), Center Back (CB), Wide Back (WB), Center Midfielder (CMD), Wide Midfielder (WMD), Attacking Midfielder (AMD) or Striker (ST). Each role requires its' taker to excel and have good stats in some features. That's why, an aggressive football player who has a fine physique and body control and who is good at tackling, marking and heading is more likely to take a defending role. For an analog reason, a footballer who has superb stamina, pace, dribbling and technique is likely to be an attacking winger.

In particular it is not uncommon to have a player who can be good in two or more different positions.
The problem here is that in the very beginning of a football career, young footballers may take some very bad choices when they try to orientate themselves leading to waste their talent and dampen their future. To be more clear and precise, in youth football academies with potentially tens of players, coaches can't try to put every player in all of the possible positions given that the number of team combinations would be astronomical. That's why some youngsters force themselves to play an attacking role like their football idol while they can easily excel in defense or are simply being forced to play in a position.

That's why our ML project can be considered as a start to a more ambitious long term project which helps youth academies coaches orientate more rapidly and efficiently the new coming players. So instead of trying the most possible number of team combinations, they can organize workshops to determine each player's abilities (pace, stamina, passing, finishing,..) and then decide enough approximately what player plays in which position.

3 Problem Definition

The dataset we ended up using contains data of 159 541 players around the world with 65 football abilities i.e features. Each feature (Pace, Stamina, Passing,..) is graded from 1 to 20 where 1 is the worst level while 20 is the highest level.
A first formulation of our problem is then :

Given the described dataset, Use ML algorithms To predict a player's position on the pitch.

Since each player be selected for different positions on the field (multi-label) and since the position for a single player can be one of the 15 positions described within the dataset; **the problem is defined as a *multi-label* and *multiclass* classification problem.**
Problem transformation:
We simplified the problem into a **one-label 7-class classification problem** : every player is assigned one position(label) among 7 possible positions, the one in which he has a maximal grade of 20.

4 Methodology

Our objectives for this project are:

(a) Collect data about m professional football players containing their best on-field positions in a vector y and their attributes for different features x1, ... , xn elated to football ; where xi represents for example Pace, Stamina, Passing, Teamwork, etc..

(b) Data Preprocessing and feature engineering.

(c) Split data into training and test sets and use classification machine learning algorithms to fit training data and predict positions of players from the test set.

(d) Evaluate and compare results of these algorithms to see which one has the biggest accuracy.

4.1 Data Collection

We explored many websites on the internet that provide football analytics and datasets about both teams and players but we didn't manage to find an "official" dataset that collects players attributes.
That's why we thought about using datasets available on video games.
We picked a dataset of players in Football Manager that we found on Kaggle.
Football Manager is a football simulation game well known for its high level of realism and the accuracy of its data.
It's even a reference used by many football clubs in real life to spot new talents and evaluate existing players instantly.
The dataset contains data of 159 541 players around the world with 65 football abilities features. Each feature (Pace, Stamina, Passing,..) is graded from 1 to 20 where 1 is the worst level while 20 is the highest level.
In Football Manager, there are around 15 different positions possible for players on the pitch.
Each player has a grade out of 20 for each position. The best position for each player has a grade equal to 20.

4.2 Data Preprocessing

A first step in data preprocessing was dealing with null values.

It appeared that in our 159 541 initial samples (i.e players) only 32 had null values in some features. Since it represents a tiny fraction of our dataset, we did not bother to fill the missing values; we simply dropped the 32 samples.

Then we tried to keep only players who played at least once for their senior or Olympic (U21) national teams mainly for the following reason: Football Manager data is more accurate and realistic with "good" and "well-known" professional players.
However, by doing so, only 16 543 players remain in our dataset. We therefore abandoned this criterion because we it causes the dataset to be reduced to approximately 1/10 of our initial data samples, which represents a huge loss (almost 90 percent of data samples is lost).

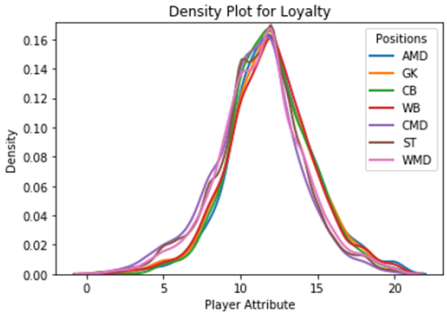
Finally we split our dataset, the remaining 159 509 samples, to :

143 092 samples **training set.**

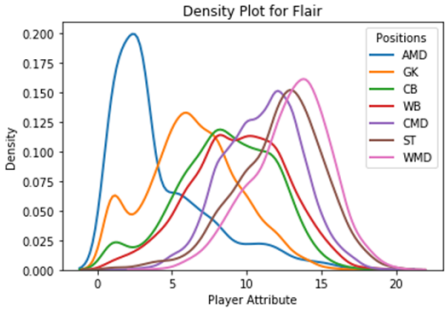
16 417 samples **test set.**

4.3 Data Visualization and Feature Selection

We tried first to select features using RFE (with logistic regression as a classifier) but the RFE kept all of the initial features. That's why we did some data visualization to manually select features that help solve the classification problem. For each of the 65 features we plotted their densities for each category of players (where one category refers to one position on the pitch). We noticed that some densities are almost identical while others are "disjoint". We can see here two different examples to illustrate our idea:
Identical densities (as with Loyalty for example) mean that probably the considered feature is universal and its values do not depend on the player's position.



(a) Loyalty.



(b) Flair.

Figure 1: Players attributes distribution.

Disjoint densities (as with Flair for example) mean that potentially this features is characteristic to a certain position on the field and thus helps do accurate classification.
We decided therefore to discard features with identical densities for the different positions which are: Concentration, Sportsmanship, Loyalty, Ambition, Adaptability, Temperament, Controversy, Pressure, Professional, InjuryProness, ImportantMatches and Determination. We kept all the other features.

Finally, we performed PCA on the new dataset to eliminate eventual data noise. We chose a threshold of 0.85 of the total variance as stopping criterion. This resulted in a dimensionality reduction from 53 dimensions to 45 principal components. The data is then projected on the new 45-dimensional sub-space.

5 Models-Learning Algorithms

In order to evaluate each learning algorithm, we estimate its test error which is a good indicator to avoid overfitting problems. We used 10-Fold CrossValidation to estimate the test error and then tuned the model's hyperparameters to control its complexity and avoid overfitting.

5.1 Decision Tree

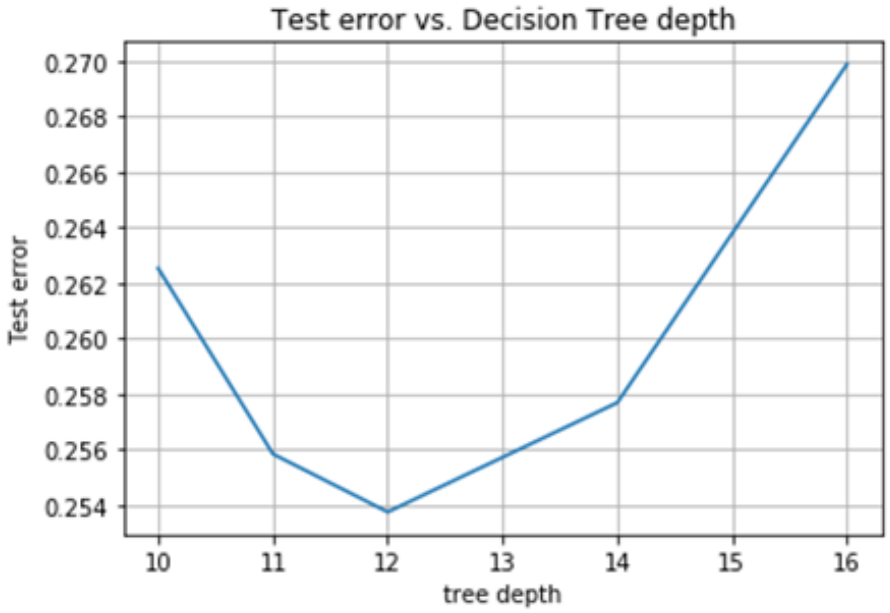


Figure 2: Test error vs Tree depth D

We used a DecisionTreeClassifier. Only "max depth" parameter of this sklearn classifier was tuned. Tree depth tuning: To determine the maximum depth parameter without overfitting our model, we used the test error as a reference. We tested the classifier for different values of the parameter "max depth" and calculated the corresponding test error using Cross-Validation. This gives us the value of the

tree depth parameter which minimizes the test error and avoids overfitting (In our example max depth =12) as shown in the following figure: .

5.2 Logistic Regression

We used the Logistic Regression Classifier implemented in scikitlearn, which we adapted to our classification problem by modifying some of its parameters:

***'lbgs'* and *'sag'* solvers which are suitable for multiclass classification. Both of these classifiers use the L2 penalty to penalize the model's complexity through parameter C.**

C represents the inverse of regularization stregth. Its default value is 1. We tried to tune this parameter trying to minimize the estimated test error by measuring the cross$_val_score$ *using both* $5-Fold$ *and* $10-FoldCrossValidation$. The minimal estimated test error (crossvalidation on the training set) is obtained with a 10-Fold crossvalidation , a parameter C=0.5 and using the 'sag' solver which is more suitable for large datasets.

The respective accuracies with a 'L2 penalty (tuned C= 0.5)' model are:

LRscore=0.82 (10-Fold CV) 'lbgs' solver.

LRscore= 0.824 (10-fold CV) with 'sag' solver.

5.3 K-Nearest Neighbors

The scikit-learn KneighborsClassifier is inherently suitable for multiclass classification problems. We therefore made use of it tuning only the hyperparameter K representing the number of nearest neighbors.

Considering the size large size of our dataset (159509 samples) and the non linear aspect of our classification problem, this learning algorithm seems to be relevant however a large hyperparamter K which increases the model's complexity. To avoid overfitting, we varied the hyperparameter K and plotted the estimated test error as shown below:
Approximately *0.8 accuracy (K=35).*

We see that the test error continues to decrease which means that at this point(K=35) we did not overfit our model, however we observed that the temporal complexity of the model increased, compared to smaller values of K and to previously tested learning algorithms. *Remark:*

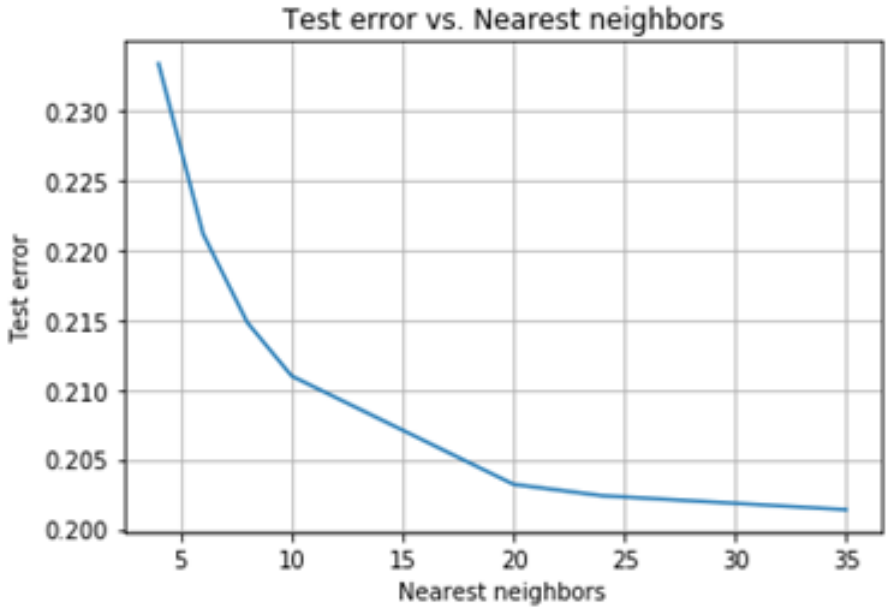


Figure 3: Test error vs Hyperparameter K

We note that the large dataset is accountable for the high complexity of this learning algorithm. Therefore, the preprocessing criterion (descriped in preprocessing section) ,which was previously abandoned, seems to be a relevant solution to this particular algorithm in order to decrease its high computational complexity O(mxsqr(n)).

6 Evaluation and Results

At this point we set the hyperparameters of each model that manimize its estimated test error i.e evaluation on training set.

We can now evaluate our chosen models on unseen data samples i.e on our test set.

For the three models, the one Logistic regression with L2 penalty is the most accurate one with testAccuracy=0.802

For the decision tree algorithm :

test error=0.276

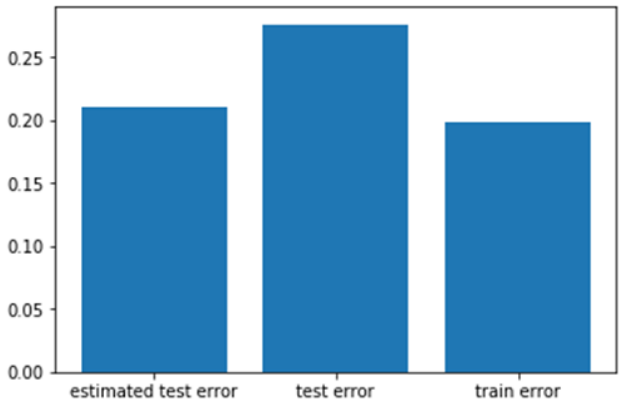
For the Logistic Regression algorithm (L2 penalty and regularization parameter C=0.5) :

test error=0.198

For the k-nearest neighbors algorithm :

test error=0.212.

To sum up below is a comparison of the error types for the three models:



error.png

Figure 4: Decision Tree errors (D=12)

error.png

Figure 5: Logistic regression errors (L2 penalty)