A Comparison of Decision Tree Classification Models on Football Player Positions

36155273, 37552384, 36261998, and 36146058

***Abstract*—A comparison of Decision Trees, Random Forests, and XGboost for classification. Each method improves on the previous in terms of accuracy but with the downside of increased computational intensity. Data was collected on football players from Europe's top 6 leagues and pre-processed to assign each player a position (GK, DF, MF, FW). The different models aimed to classify the players on their position and a comparison of the results was completed to evidence the differences in their classification scores and computational efficiency.**

I. Introduction

In this report, three different closely related models were applied: decision trees, random forests, and XGboost. Each method can be theoretically seen as an improvement on the previous in terms of generalisation and accuracy. This report aims to show a comparison of the different algorithms by applying them to the same chosen dataset. This dataset gave hundreds of different statistics on football players in Europe's top 6 leagues and most importantly contained the position they played on the pitch. This was used as the target feature for classification and the aim was to explore the differences in the three model's predictions and how the decision trees were created, focusing on the key features that explained decisions. The report will go into detail on the methodology behind the algorithms, explore the results of classification and compare the performance of the models to evidence their known differences on real world data.

II. Methods

A. Decision Trees

A decision tree is a supervised learning method that uses branches to demonstrate every possible outcome. They are used for both regression trees and classification trees. This report will only look at classification trees, as the data used has several classes for us to classify between. A classification tree aims to partition covariates into distinct regions. The prediction is performed locally in each of these regions and then involves seeing which regions the data point belongs to. The method starts with all the data points and chooses the best line to split the data into two regions. To choose the best line it looks at which line maximally reduces the impurity measure at each decision node. In this case, the measure of impurity is the GINI Index. The GINI Index takes the form

$$f_{GINI}(p) := p(1-p) \tag{1}$$

Where $f_{GINI}(0) = f_{GINI}(1) = 0$. It is a 'greedy' algorithm and so aims to take the best possible split of the data at every iteration. These decision trees, as they continue to partition the data until just one observation is left, end up massively overfitting the data. To prevent this either: introduce a complexity parameter α in order to penalise more complex models or limit the maximum depth of the tree to ensure the model does not end up too complex.

B. Random Forests

A random forest is, as stated by Breiman (2001), a classifier that consists of a group of tree-structured classifiers $h(\boldsymbol{x}, \Theta_k, k = 1, \dots$ where the Θ_k are i.i.d random vectors and each tree casts a unit vote for the most popular class at input $\boldsymbol{x}$. The method is similar to bagging. Bagging is where multiple data sets are generated from the original, large sets of classifiers on each of these data sets are grown, and finally the performance of these are compiled to achieve a final single prediction. Random forests follow the same procedure, however, in a random forest the covariates are also subsampled at each decision node to reduce the correlation between each tree. Therefore, the procedure for random forests is: to create bootstrap samples $\tilde{X}^{(t)}$ for $t = 1, \ldots, T$; begin to create trees for each sample; at each decision node subsample the number of covariates in the split data (keeping $q < p$ covariates for the next split); build the trees using the subsampled set; and use an average combiner to compile the predictions.

C. Extreme Gradient Boosted Decision Trees

Schapire (2003) describes boosting as a method of accurate prediction by combining many relatively inaccurate predictions, in other words improving a single model by fitting many models and combining them (Elith et al. 2008). Boosting works similarly to bagging in that it combines many predictions together to get a more accurate prediction. However, unlike bagging, boosting uses all the previous predictions to construct the next prediction (James et al. 2017). Therefore, a boosted decision tree is constructed by taking lots of smaller decision trees and combining the results. These decision trees can be very small, for example just a few terminal nodes. The smaller decision trees we form are from the residuals from the model. So, for each new tree formed, the current residuals are used instead of the outcome. The new decision tree formed is added to the fitted function so that the residuals are updated, and then the new residuals will be used for the next decision tree. By using these small decision trees, areas of the boosted decision tree that are least accurate are slowly improved. This reduces overfitting and increases accuracy.

The following method described for XGBoosted trees is in parallel with existing literatures in gradient boosting such as Chen and Guestrin (2016). The second order approximation concept is from Friedman et al. (2000).

Gradient boosted trees (XGBoost) uses an ensemble decision tree model. This is a model that includes multiple classification and regression trees. The model predicts the output, $\hat{y}_i$, using K additive functions

$$\hat{y}_i = \psi(\boldsymbol{x}_i = \sum_{k=1}^{K} f_k(\boldsymbol{x_i}, f_k \in F$$

Where $F = \{f(\boldsymbol{x}) = w_{q(\boldsymbol{x})}\}$, $q : \mathbf{R}^m \to T$ and $w \in \mathbf{R}^T$. Here w is the weight of each leaf, f_k is each of the trees built by the decision tree ensemble, q is the structure of each independent tree, T is the number of leaves within a tree and m is the number of features of the original data set. An objective function is then created that we aim to minimise. This objective function consists of two parts: a loss function, l and a complexity parameter, Ω.

$$L(\psi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

For gradient boosted decision trees, the complexity parameter is

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

Since the objective function for the tree ensemble model has parameters that are functions, it cannot be optimized using traditional optimization methods and must instead be optimized in an additive manner. Let $\hat{y}_i^{(t)}$ be the i-th instance of the t-th iteration, then the new objective function to be minimised is

$$L^{(t)} = \sum_{i=1}^{n} l(\hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i), y_i) + \Omega(f_t)$$

In other words the f_t is added that gives us the biggest improvement in the model, this is known as being greedy. Generally, the objective function is then optimized using second order approximation as follows

$$L^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$. This can be simplified by removing all constants to give the t-th iteration

$$\tilde{L}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \Omega(f_t)$$

If the complexity parameter is subbed in, this function can then be arranged to give the optimal weight of a leaf for a fixed structure $q(\boldsymbol{x})$

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

and the optimal value for this can be calculated using

$$\tilde{L}^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j}, g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda}$$

This can be used similar to the impurity measure in normal decision trees to score the model. A greedy algorithm is then used to add a branch, at each iteration i, to the tree based on which trees provide the best scores in the equation above.

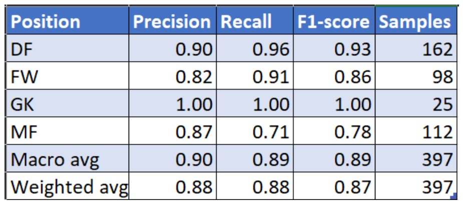
This method is more efficient and more accurate than normal decision tree classifiers. This is because it uses processes such as parallel processing and uses decision tree ensembles to effectively use all resources and reduce overfitting. It also has many in built hidden parameters, such as learning rate, that can be used to tune any XGBoosted model to provide more suitable results. XGBoost, however, is a very complex system therefore can cause increased computation time.

III. Data Pre-Processing

The dataset was found on Kaggle, containing 143 features on almost 3000 different football players from Europe's top 6 leagues for the 2021/22 season but obviously the data needed to be pre-processed before applying the models. To make the statistics representative, players that had less than 500 minutes in the season were removed, leaving a total of 1983 players. To simplify classification and increase interpretability, only four possible positions were given to players based off the first and most preferred position recorded in the data (GK, DF, MF, FW). The data was quite equally spread between classes which is preferable and gives generally increased performance on test data. Furthermore, some further features were removed due to not being relevant to in game performance or being extremely correlated with other features (e.g. passes completed and touches on the ball). It is interesting to note that the data does not be standardised or normalised before applying the models because the calculations involved in decision trees (GINI impurity) are not distance-based metrics and are simply using the proportions of labels in the data. The final dataset did not have any null values and contained 98 features to be used to classify the player positions.

IV. Experiment/Results

The results of those three models can be seen by executing the classification report which is a summary of each model. This report includes the accuracy, precision, recall, and F1-Score for each class of the classification model. The training time of each model was also recorded.



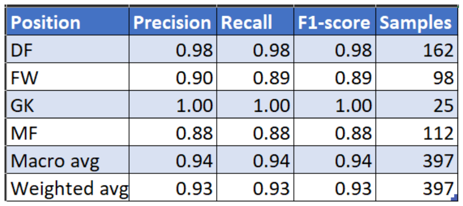
Position	Precision	Recall	F1-score	Samples
DF	0.90	0.96	0.93	162
FW	0.82	0.91	0.86	98
GK	1.00	1.00	1.00	25
MF	0.87	0.71	0.78	112
Macro avg	0.90	0.89	0.89	397
Weighted avg	0.88	0.88	0.87	397

Fig. 1: Decision Tree Classification Scores

A. *Normal Decision Tree*

For this model the overall accuracy is 88%. The precision scores for the classes are high. This means that the model is not prone to false positives. However, the recall scores are lower in FW and MF classes meaning that the model struggles to identify certain instances of those classes. Overall, having in mind the factor of the report, the Goalkeepers class is the best performing class of the model as the model can identify the GKs better than the other three classes (highest precision, recall and F1-score with the score of 100%). Then the defenders-DF class is the next best performing class as its precision (90%), recall (96%) and f1-score (93%) are the second highest. After, the third best is the forwards-FW class with precision 82%, recall 91% and f1-score 86%. Finally, the hardest class to predict is midfielders as their precision (87%), recall (71%) and f1-score (78%) are the lowest meaning that the model finds it hard to identify the midfielders in this model. The time taken for this model to train is 0.67s.

B. *Random Forest Decision Tree*

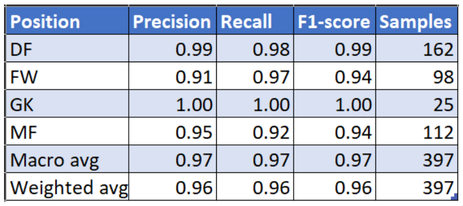


Position	Precision	Recall	F1-score	Samples
DF	0.98	0.98	0.98	162
FW	0.90	0.89	0.89	98
GK	1.00	1.00	1.00	25
MF	0.88	0.88	0.88	112
Macro avg	0.94	0.94	0.94	397
Weighted avg	0.93	0.93	0.93	397

Fig. 2: Random Forest Classification Scores

For this model, the overall accuracy is 93%. The precision scores (can be found in figure ?) for the classes are high. This means that the model is not prone to false positives. However, the recall scores are lower in MF and FW classes meaning that the model struggles to identify certain instances of those classes. Overall, having in mind the factor of the report, the Goalkeepers class is the best performing class of the model as the model can identify the GKs better than the other three classes (highest precision, recall and F1-score with the score of 100%). Then the defenders-DF class is the next best performing class as its precision (98%), recall (98%) and f1-score (98%) are the second highest. After, the third best is the forwards-FW class with precision 90%, recall 89% and f1-score 89%. Finally, the hardest class to predict is midfielders as their precision (88%), recall (88%) and f1-score (88%) are the lowest meaning that the model finds it hard to identify the midfielders in this model. The time taken for this model to train is 0.92s.

C. *XGBoosted Decision Tree*



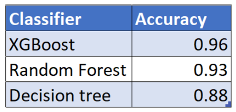
Position	Precision	Recall	F1-score	Samples
DF	0.99	0.98	0.99	162
FW	0.91	0.97	0.94	98
GK	1.00	1.00	1.00	25
MF	0.95	0.92	0.94	112
Macro avg	0.97	0.97	0.97	397
Weighted avg	0.96	0.96	0.96	397

Fig. 3: XGboost Classification Scores

For this model, the overall accuracy is 96%. The precision scores for the classes are high. This means that the model is not prone to false positives. However, the recall scores are lower in MF and FW classes meaning that the model struggles to identify certain instances of those classes. Overall, having in mind the factor of the report, the Goalkeepers class is the best performing class of the model as the model can identify the GKs better than the other three classes (highest precision, recall and F1-score with the score of 100%). Then the defenders-DF class is the next best performing class as its precision (99%), recall (98%) and f1-score (99%) are the second highest. After, the third best is the midfielders-MF class with precision 95%, recall 92% and f1-score 94%. Finally, the hardest class to predict is forwards-FW as their precision (91%), recall (97%) and f1-score (94%) are the lowest meaning that the model finds it hard to identify the midfielders in this model. The time taken for this model to train is 2.47s.

V. Conclusion

Based on the classification report scores of the three methods used in this study - Decision Trees, XGboosted Decision Trees, and Random Forest Decision Trees - we can determine which method was the best for the dataset. It can be observed

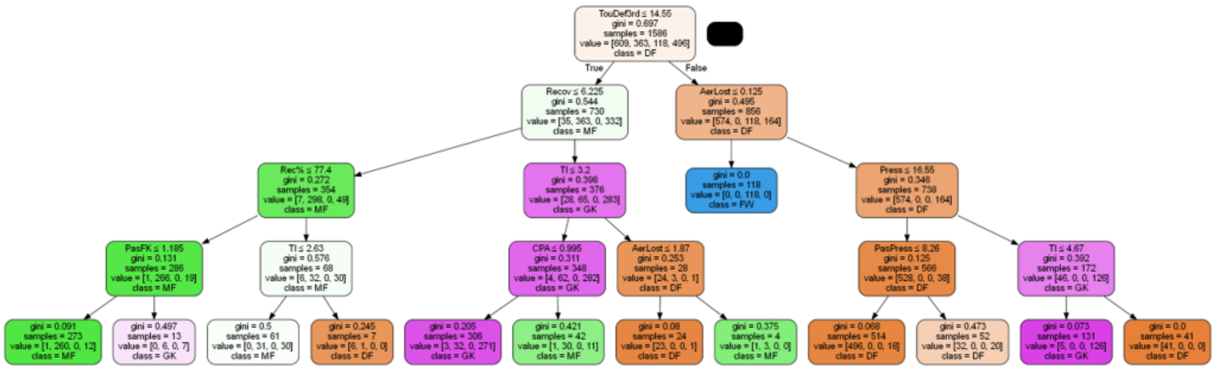


Classifier	Accuracy
XGBoost	0.96
Random Forest	0.93
Decision tree	0.88

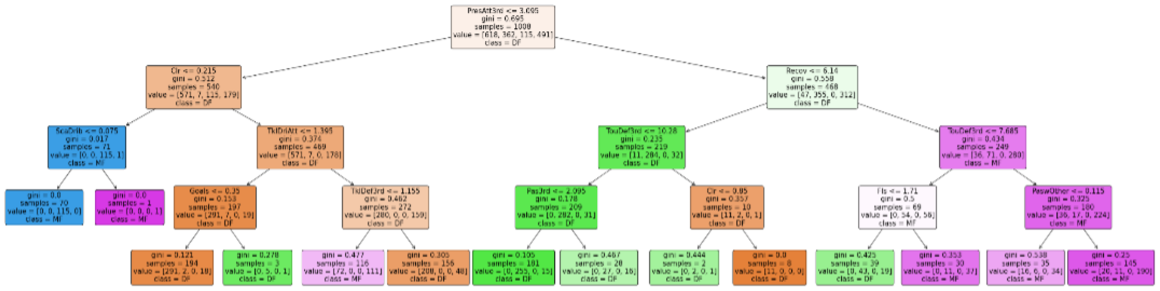
Fig. 4: Accuracy of each model

that the Decision Trees model has an accuracy score of 88%, compared to 93% for the Random Forest model and 96% for the XGboosted model. The computational efficiency of the three models is also worth noting, with the XGboosted model taking much longer to train (2.47 seconds), the Decision Trees model taking 0.67 seconds, and the Random Forest model taking 0.92 seconds to fit the models and plot the trees. Based on these results, we conclude that the XGboosted model is the best performer in terms of accuracy, followed by the Random Forest model. All three models demonstrated high accuracy and could be applied to similar datasets with a high degree of success. The results have successfully shown evidence of the differences between the models. The plots for all the methods were limited to a maximum depth of 4 in order to make them more readable and make the results more directly comparable, as seen in appendices A, B and C. The nodes show the decisions made to split the data at each branch, so the classification path of the data can be intuitively followed by anyone. This shows which features are most important in terms of the player's position and is a huge advantage of the decision tree classifiers over other methods. Positional factors such as touches and presses in different thirds of the pitch seem key in the best splits of the data for the decision tree and random forest, whereas the XGboosted tree interestingly focuses on more specific stats such as clearances, throw-ins and number of presses. The difference in approach from this algorithm is undoubtedly due to it's methodology and not taking a 'greedy' approach to learning. XGboost considers previous information to build upon weaker parts of the tree and as a result, more unique features such as throw-ins become very important in classification. Overall, the differences in methodologies between algorithms has been shown through the distinct classification scores, computational efficiencies, and the choice of feature splits in the trees.

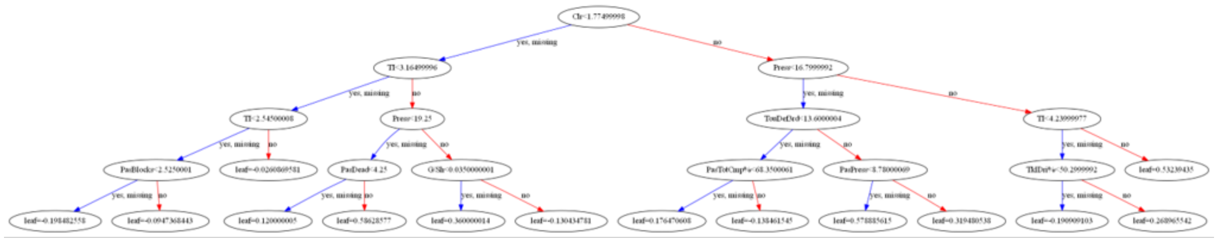
Appendix A
Decision Tree


TouDef3rd ≤ 14.55 gini = 0.697 samples = 1586 value = [609, 363, 118, 496] class = DF
[redacted]
True
False
Recov ≤ 6.225 gini = 0.544 samples = 730 value = [35, 363, 0, 332] class = MF
AerLost ≤ 0.125 gini = 0.495 samples = 856 value = [574, 0, 118, 164] class = DF
Rec% ≤ 77.4 gini = 0.272 samples = 354 value = [7, 298, 0, 49] class = MF
TI ≤ 3.2 gini = 0.398 samples = 376 value = [28, 65, 0, 283] class = GK
gini = 0.0 samples = 118 value = [0, 0, 118, 0] class = FW
Press ≤ 16.55 gini = 0.346 samples = 738 value = [574, 0, 0, 164] class = DF
PasFK ≤ 1.185 gini = 0.131 samples = 286 value = [1, 266, 0, 19] class = MF
TI ≤ 2.63 gini = 0.576 samples = 68 value = [6, 32, 0, 30] class = MF
CPA ≤ 0.995 gini = 0.311 samples = 348 value = [4, 62, 0, 282] class = GK
AerLost ≤ 1.87 gini = 0.253 samples = 28 value = [24, 3, 0, 1] class = DF
PasPress ≤ 8.26 gini = 0.125 samples = 566 value = [528, 0, 0, 38] class = DF
TI ≤ 4.67 gini = 0.392 samples = 172 value = [46, 0, 0, 126] class = GK
gini = 0.091 samples = 273 value = [1, 260, 0, 12] class = MF
gini = 0.497 samples = 13 value = [0, 6, 0, 7] class = GK
gini = 0.5 samples = 61 value = [0, 31, 0, 30] class = MF
gini = 0.245 samples = 7 value = [6, 1, 0, 0] class = DF
gini = 0.205 samples = 306 value = [3, 32, 0, 271] class = GK
gini = 0.421 samples = 42 value = [1, 30, 0, 11] class = MF
gini = 0.08 samples = 24 value = [23, 0, 0, 1] class = DF
gini = 0.375 samples = 4 value = [1, 3, 0, 0] class = MF
gini = 0.068 samples = 514 value = [496, 0, 0, 18] class = DF
gini = 0.473 samples = 52 value = [32, 0, 0, 20] class = DF
gini = 0.073 samples = 131 value = [5, 0, 0, 126] class = GK
gini = 0.0 samples = 41 value = [41, 0, 0, 0] class = DF

Appendix B
Random Forest


PresAtt3rd <= 3.095 gini = 0.695 samples = 1000 value = [618, 362, 115, 491] class = DF
Clr <= 0.215 gini = 0.512 samples = 540 value = [571, 7, 115, 179] class = DF
Recov <= 6.14 gini = 0.558 samples = 468 value = [47, 355, 0, 312] class = DF
ScaDrib <= 0.075 gini = 0.017 samples = 71 value = [0, 0, 115, 1] class = MF
TklDriAtt <= 1.395 gini = 0.374 samples = 469 value = [571, 7, 0, 178] class = DF
TouDef3rd <= 10.28 gini = 0.235 samples = 219 value = [11, 284, 0, 32] class = DF
TouDef3rd <= 7.685 gini = 0.434 samples = 249 value = [36, 71, 0, 280] class = MF
gini = 0.0 samples = 70 value = [0, 0, 115, 0] class = MF
gini = 0.0 samples = 1 value = [0, 0, 0, 1] class = MF
Goals <= 0.35 gini = 0.153 samples = 197 value = [291, 7, 0, 19] class = DF
TklDef3rd <= 1.155 gini = 0.462 samples = 272 value = [280, 0, 0, 159] class = DF
PasJrd <= 2.095 gini = 0.178 samples = 209 value = [0, 282, 0, 31] class = DF
Clr <= 0.85 gini = 0.357 samples = 10 value = [11, 2, 0, 1] class = DF
Fls <= 1.71 gini = 0.5 samples = 69 value = [0, 54, 0, 58] class = MF
PaswOther <= 0.115 gini = 0.325 samples = 180 value = [36, 17, 0, 224] class = MF
gini = 0.121 samples = 194 value = [291, 2, 0, 18] class = DF
gini = 0.278 samples = 3 value = [0, 5, 0, 1] class = DF
gini = 0.477 samples = 116 value = [72, 0, 0, 111] class = MF
gini = 0.305 samples = 156 value = [208, 0, 0, 48] class = DF
gini = 0.105 samples = 181 value = [0, 255, 0, 15] class = DF
gini = 0.467 samples = 28 value = [0, 27, 0, 16] class = DF
gini = 0.444 samples = 2 value = [0, 2, 0, 1] class = DF
gini = 0.0 samples = 8 value = [11, 0, 0, 0] class = DF
gini = 0.425 samples = 39 value = [0, 43, 0, 19] class = DF
gini = 0.353 samples = 30 value = [0, 11, 0, 37] class = MF
gini = 0.538 samples = 35 value = [16, 6, 0, 34] class = MF
gini = 0.25 samples = 145 value = [20, 11, 0, 190] class = MF

Appendix C
XGBoosted Decision Tree



A References

1) Breiman, L., 2001, ”Random Forests”. Machine Learning 45, (pp 5–32) . https://doi.org/10.1023/A:1010933404324
2) Chen, T., and Guestrin, C., “XGBoost: A Scalable Tree Boosting System,” ACM Knowledge Discovery in Data, (pp. 785-794).
3) Elith, J., Leathwick, J. R. and Hastie, T., 2008, ”A working guide to boosted regression trees”, Journal of Animal Ecology, Volume 77, Number 4. (pp. 802-813)
4) James, G., D. Witten, T. Hastie, and R. Tibshirani. 2017. ”An Introduction to Statistical Learning with Applications in R”. Springer. (pp 345-352). https://www-bcf.usc.edu/ gareth/ISL/
5) Kaggle Dataset, ”2021-2022 Football Player Stats.” [Online]. Available: https://www.kaggle.com/datasets/vivovinco/20212022-football-player-stats
6) Schapire, R. E., 2003, ”The Boosting Approach to Machine Learning: An Overview”, Nonlinear Estimation and Classification
7) XGBoost, “Introduction to Boosted Trees.” [Online]. Available: https://xgboost.readthedocs.io/en/latest/tutorials/model.html