Supervised Learning: Classification
FIFA 18

Snorri Bjarkason

TABLE OF CONTENTS

Understanding FIFA Player Data and Attribute Ratings *4*

Analyzing the Dataset *6*

Player Position Distribution and Dataset Overview **7**

Visualizing Player Attributes **8**

Process *16*

Data Preprocessing **16**

Model Selection **18**

K-Nearest Neighbors (KNN) 18

Support Vector Machine (SVM) 18

Decision Trees 18

Random Forest 19

Gradient Boosting 19

Naive Bayes 19

Justification of Model Choices **20**

Justification of hyperparameters choices **20**

Training Time **23**

Results *23*

KNN **24**

SVM **28**

Decision Tree **33**

Random Forrest **38**

Gradiant Boosting **42**

Naïve Bayes **49**

Conclusions *51*

Interpretation of Results **51**

Comparison with the Literature **52**

Importance of Hyperparameter Tuning **53**

Conclusion **53**

Future Work *54*

References *55*

In football, being able to predict what position a player is best suited for can be very helpful. Teams use data and statistics to make decisions about which players to sign, how to organize their squads, and who to play in certain positions. This project focuses on building a machine learning model to predict whether a football player is a forward, midfielder, defender, or winger based on their in game statistics.

The data for this project comes from the FIFA 21 Complete Player Dataset1, which includes detailed information on player attributes like pace, shooting, passing, dribbling, defending, and physicality. These stats are used as features, and the players position is the target we are trying to predict. With thousands of players included, this dataset is large enough to provide meaningful insights.

We will test several machine learning models to determine which works best for this problem, including k-nearest neighbors (KNN), support vector machines (SVM), decision trees, random forests, gradient boosting and naive nayes. By using techniques like cross validation and grid search, we'll also tune the models to ensure they perform as well as possible.

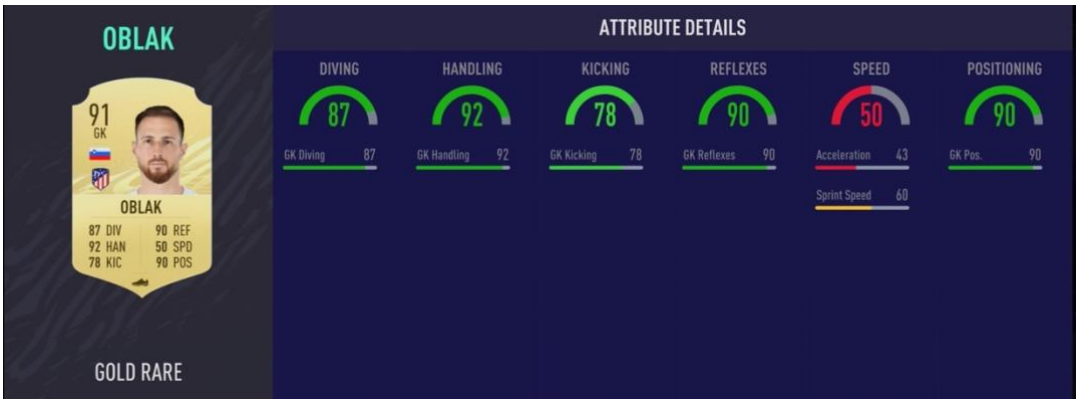
This report will compare how each model performs and explain which one works best for predicting player positions. The results could help improve how teams use data in making decisions about players.

Understanding FIFA Player Data and Attribute Ratings

Before diving into the process, it is essential to understand the context of the data used in this project: FIFA player data. FIFA, developed by EA Sports, is one of the most popular football video games worldwide. In FIFA, players are rated based on their real world performances in the previous football season. Each player is given an overall rating, along with specific ratings across various skill categories that reflect their strengths and weaknesses on the pitch. These categories include pace, shooting, passing, dribbling, defending, and physicality. And within for example the pace category, players are evaluated on both acceleration and sprint speed. This multi dimensional approach to player ratings allows for a detailed understanding of a player's abilities. See an example of a FIFA rating in the photo below:

[1] (Leone *FIFA 21 complete player dataset*)

The data used for this project focuses on outfield players, namely forwards, midfielders, and defenders. While goalkeepers are a part of FIFA as well, they follow a completely different rating system tailored to their role on the field. Their performance is measured by specific metrics such as reflexes, handling, diving, and positioning, which are not relevant for outfield players. Therefore, goalkeepers have been excluded from this project to maintain consistency in the classification process, focusing only on players who share common categories like pace, shooting, and defending.



The FIFA dataset is particularly well suited for this project due to the inherent diversity among football players. No two players are the same, even if they play in similar positions. For instance, while wingers are often characterized by their speed, not every winger relies on pace, some may excel in dribbling or passing instead. Similarly, midfielders can vary greatly, some are defensively solid but lack shooting ability, while others may be more attack focused with excellent shooting skills but weaker defensive attributes. This diversity makes FIFA player data a rich and challenging dataset for classification tasks, as it requires the model to capture these nuances to make accurate predictions. This project leverages this complexity to classify football players into broad positional categories based on their in game attributes, such as whether a player is a forward, midfielder, or defender. This example highlights how players in the same position can have different stats. Even though they play the same role on the field, their attributes can vary, showing that players bring unique skills and strengths to their position as this example provides:

Analyzing the Dataset

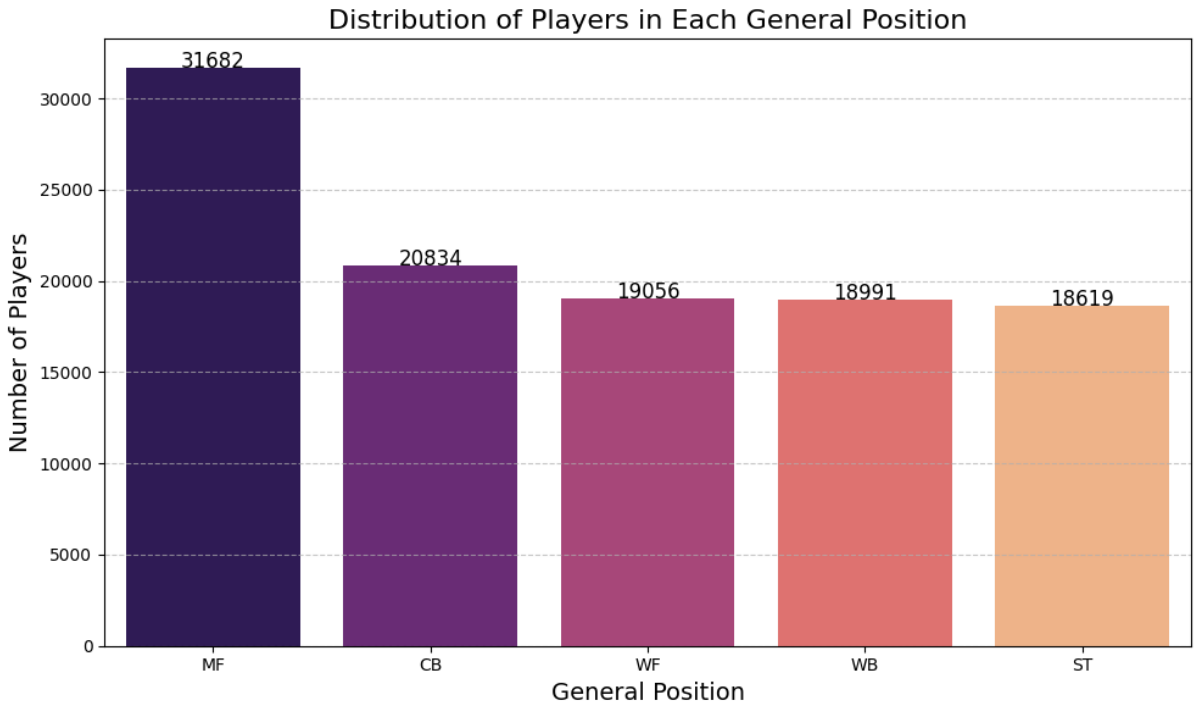
Also it is important before we dive into building machine learning models, that first we analyze the data at our disposal. By exploring the attributes of the players in our dataset, we can gain valuable insights that will guide our model building process. Visualizing the distributions and relationships between different player attributes can help us understand how various positions are characterized in the game of FIFA.

In FIFA, players are defined by attributes (pace, shooting, passing..), and these attributes vary significantly depending on their playing position. For example, forwards might be expected to have high shooting attributes, as their primary role is to score goals. Similarly, wingers could possess a high level of dribbling and pace, allowing them to navigate past defenders and create scoring opportunities. On the other hand, defenders would logically have higher defending statistics, as their role focuses on preventing goals.

Through this initial exploration, we hope to confirm or challenge these expectations by visualizing the data in meaningful ways. This analysis can potentially offer us important clues as to which features (attributes) will be most useful when it comes to building accurate classification models.

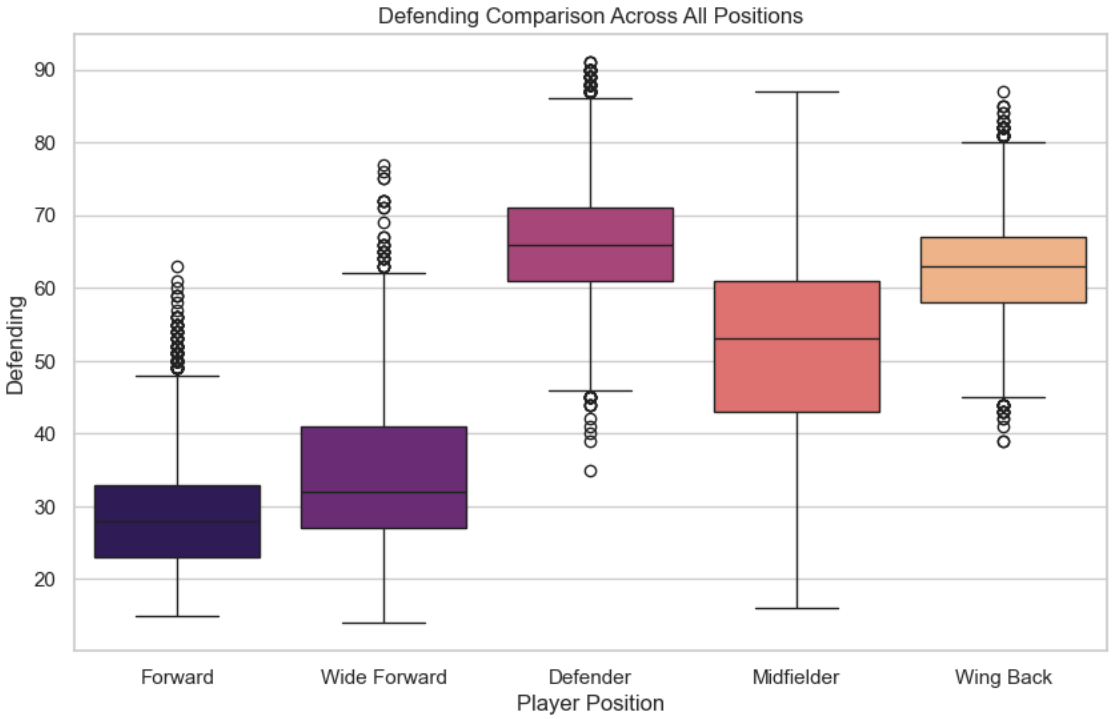
PLAYER POSITION DISTRIBUTION AND DATASET OVERVIEW

In the below bar plot, we see the distribution of players across five general positions: midfielders (MF), center backs (CB), wide forwards (WF), wing backs (WB), and strikers (ST). The dataset contains 90,182 players, with the majority being classified as midfielders (31,682). This is not surprising given that the midfielder category is broad and includes central midfielders, attacking midfielders, and defensive midfielders (CM, CAM, and CDM). Center backs (CB) make up the second largest group with 20,834 players, while the remaining positions, wide forwards (19,056), wing backs (18,991), and strikers (18,619) are more evenly distributed.



VISUALIZING PLAYER ATTRIBUTES

Below, we will generate a series of visualizations to examine key attributes across the different player positions. These graphs will provide a foundation for understanding the variations in player roles and their statistical characteristics.

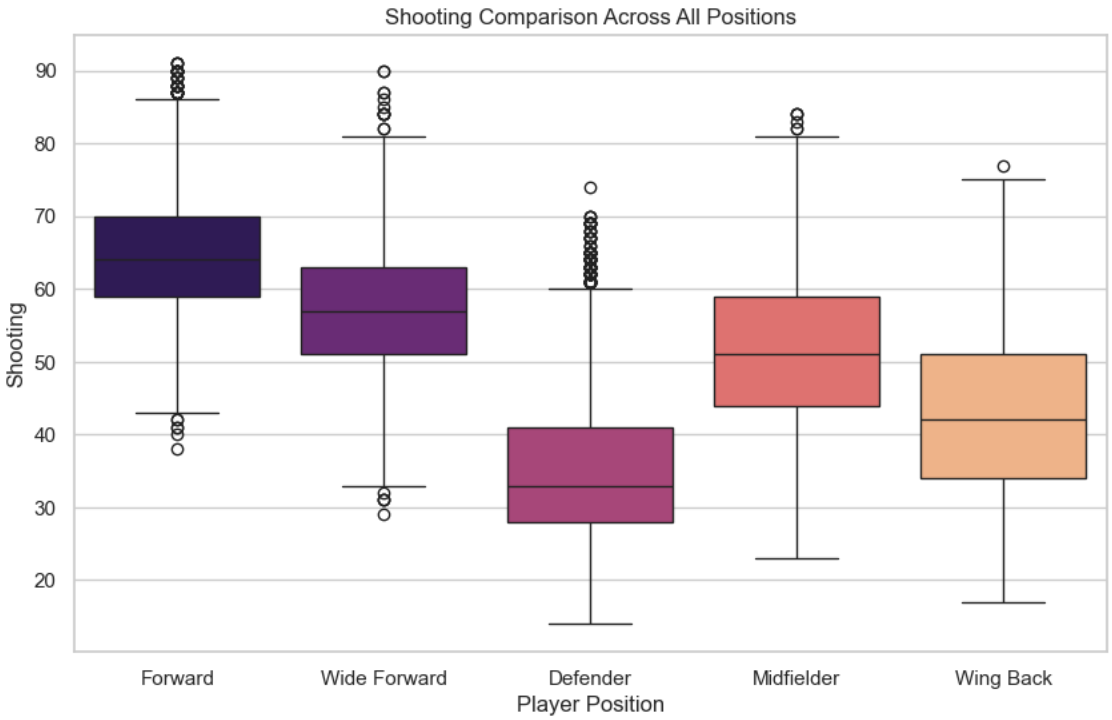


The graph above compares defending attributes across player positions, showing clear distinctions based on role. As expected, defenders have the highest defending ratings, with most scores clustering between 60 and 70, which aligns with their primary role of preventing goals.

Wing backs also display strong defensive skills, though slightly lower than central defenders, reflecting their dual role of defending and contributing to attacks. Midfielders have a broader range of defending scores, typically between 58 and 67, as some focus on defensive tasks while others lean toward offensive play.

Forwards and wide forwards predictably show the lowest defending ratings, with most scores between 20 and 40, as their primary responsibility is scoring and creating chances.

In summary, this plot confirms that defending abilities align with player roles, defenders and wing backs excel, while forwards and wide forwards have lower ratings. Midfielders show more variation due to their diverse responsibilities on the field.

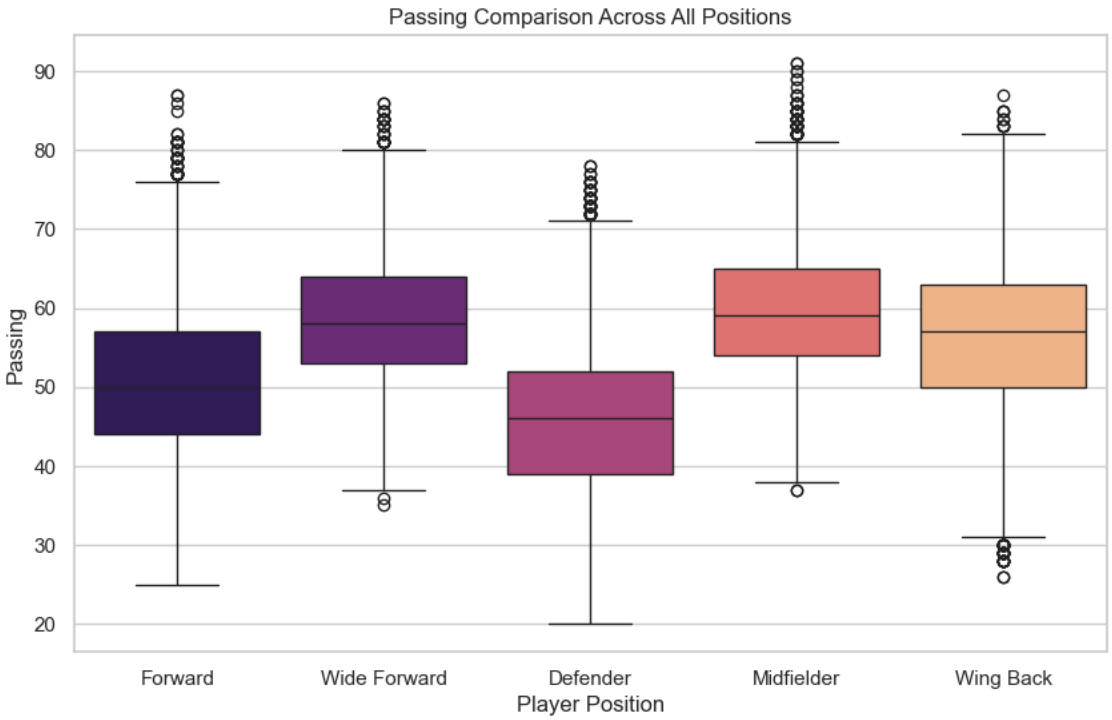


The above graph compares shooting abilities across different player positions, showing clear trends based on role. Forwards naturally have the highest shooting ratings, with most scores between 60 and 70, reflecting their primary job of scoring goals. Wide forwards also demonstrate strong shooting skills, though slightly lower than central forwards, aligning with their role of creating and scoring chances from wider areas.

Midfielders show moderate shooting abilities, typically ranging from 45 to 60, as some midfielders have offensive duties, while others focus more on playmaking and defense. Defenders have the lowest shooting scores, which makes sense given their primary defensive role. The majority of defender scores cluster between 30 and 40.

Wing backs fall between midfielders and defenders, with shooting scores mostly ranging from 35 to 50, indicating they occasionally contribute to attacking plays but are not relied on for scoring.

In summary, this plot highlights that shooting ability aligns well with the primary role of each position, forwards excel, midfielders show moderate shooting capabilities, and defenders lag behind, focusing more on defensive duties.



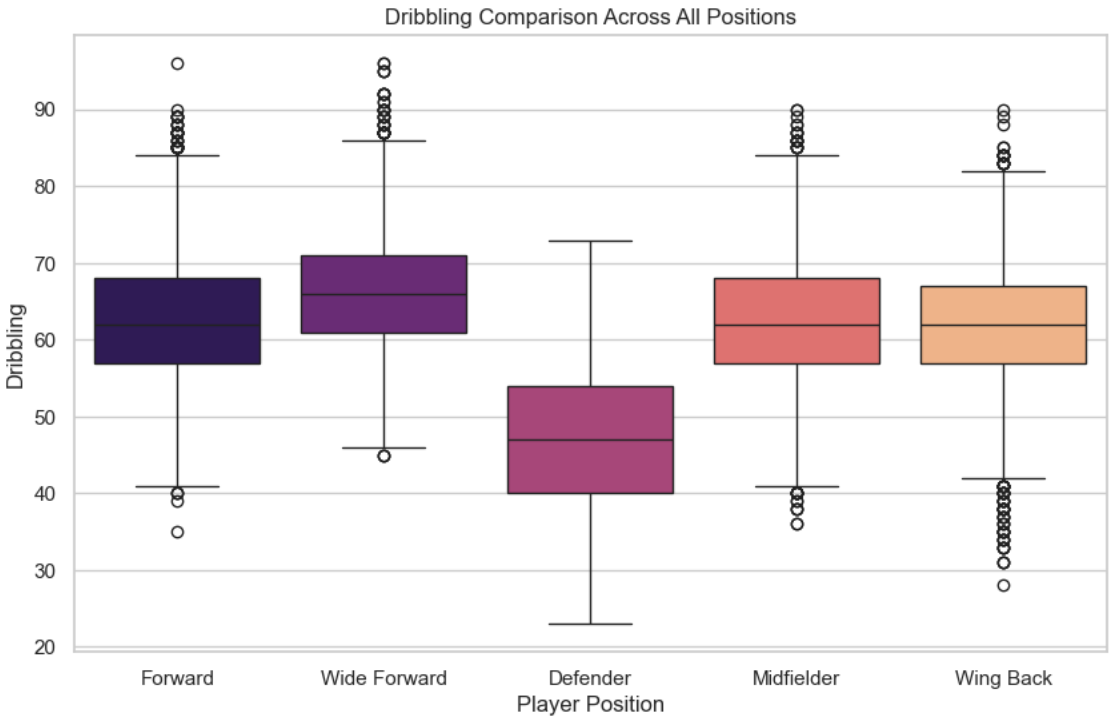
The graph highlights the passing abilities across different player positions. Midfielders stand out with the highest passing scores, mostly ranging from 55 to 65. This reflects their primary role in distributing the ball and maintaining possession, making passing a key skill in their gameplay.

Wide forwards also show strong passing abilities, with most scores falling between 53 and 64, which aligns with their role in creating chances and crossing the ball into the box. Similarly, forwards demonstrate moderate passing skills, with a comparable range to wide forwards. Their passing ability helps them link up with other attackers during offensive plays.

Wing backs exhibit respectable passing abilities, mostly between 50 and 63, as their modern role requires them to contribute both defensively and offensively. They often participate in build up play, making passing an important aspect of their game.

Defenders, as expected, have the lowest passing scores, generally between 40 and 52. Their focus is on defensive duties, so passing is not as crucial for their role compared to other positions.

In conclusion, passing is essential for midfielders and wide forwards, while forwards and wing backs still require a decent level of passing ability. Defenders, however, prioritize other attributes over passing.



The dribbling comparison graph highlights the varying dribbling abilities across player position. Wide forwards show the highest dribbling abilities overall, with scores concentrated between 60 and 70, and several players achieving scores above 90. This makes sense, as wide forwards often rely on their dribbling skills to beat defenders on the wings and create goal scoring opportunities.

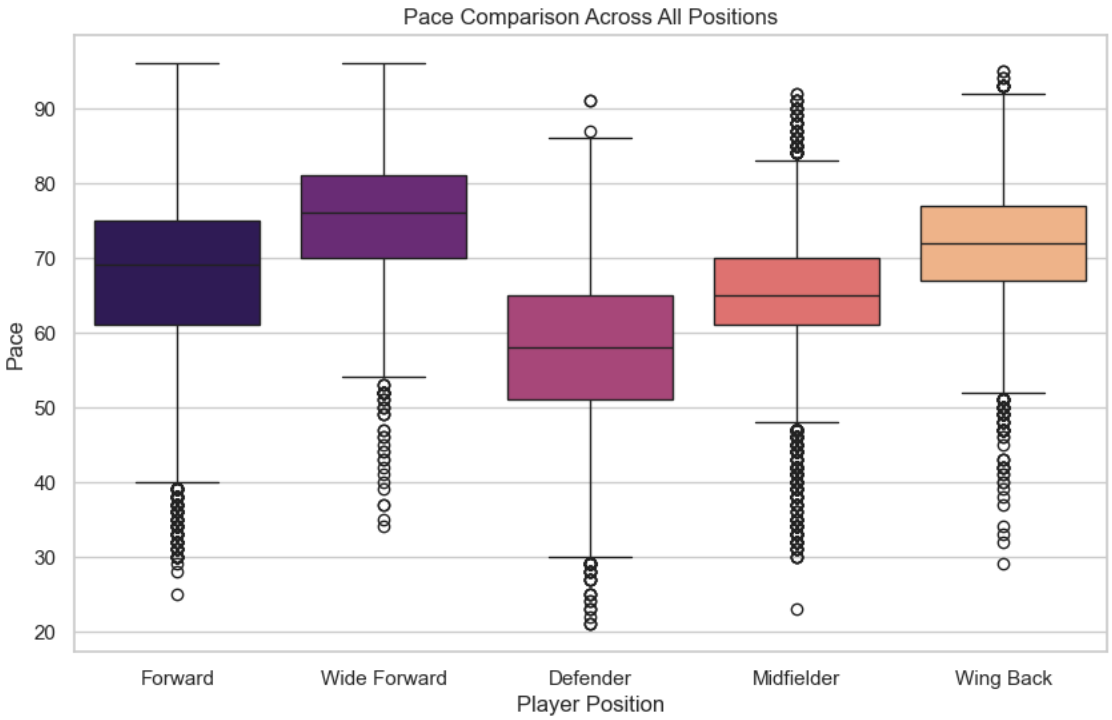
Forwards also show strong dribbling abilities, with a similar distribution to wide forwards, **albeit** with slightly more variation. Forwards need dribbling skills to navigate through tight defenses, but they might focus more on positioning and shooting compared to wide forwards.

Midfielders come next in terms of dribbling abilities, with scores primarily ranging between 55 and 65. This is expected, as midfielders often play a crucial role in maintaining possession and moving the ball forward. Their dribbling skills allow them to escape pressure and transition the play effectively.

Wing backs have a similar dribbling range to midfielders, which reflects their modern role in advancing up the flanks and contributing to the attack. Wing backs, especially in more attacking formations, often need dribbling skills to progress the ball or provide crosses.

As expected, defenders have the lowest dribbling scores, with most players in this position falling between 40 and 55. Defenders focus more on stopping opposition attacks rather than carrying the ball forward, so dribbling is less of a priority for them.

In summary, dribbling is most critical for wide forwards and forwards, followed closely by midfielders and wing backs, while defenders display the least need for advanced dribbling abilities.



The graph above highlights the pace comparison across different player positions, with wide forwards clearly excelling in this attribute. Most wide forwards have pace scores clustered between 70 and 80, with many even reaching 90. This makes sense, as wide forwards frequently rely on their speed to beat defenders and create scoring opportunities from the flanks.

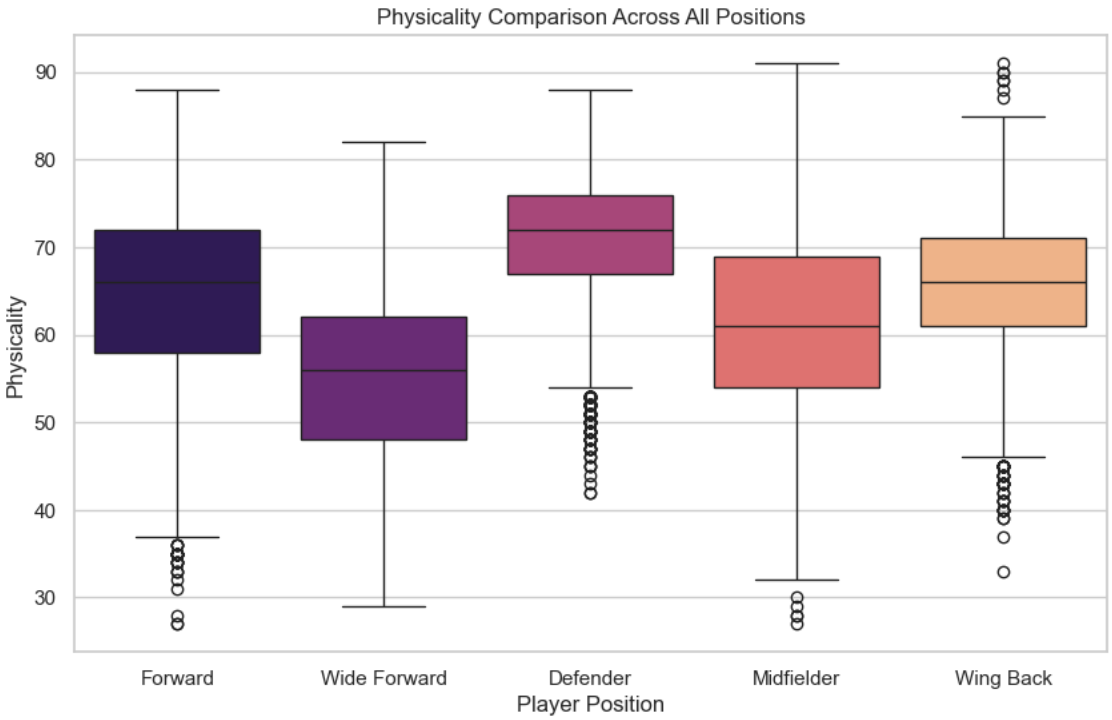
Forwards also exhibit strong pace, although their range shows slightly more variation compared to wide forwards, with the majority scoring between 60 and 75. Pace is crucial for forwards, especially when breaking through defensive lines, but they may focus more on finishing and positioning compared to wide forwards.

Wing backs, reflecting their dual role in defense and attack, also demonstrate high levels of pace, with scores primarily between 67 and 77. Modern wing backs often sprint up and down the field, making pace an important asset.

Midfielders display a slightly lower pace range, typically between 60 and 70. Although pace is not a primary skill for midfielders, it is still useful when transitioning play or covering ground during defensive duties.

Lastly, defenders predictably show the lowest pace scores, ranging mostly between 50 and 65. While pace is not a critical requirement for defenders, it can still be advantageous when tracking back against fast attackers.

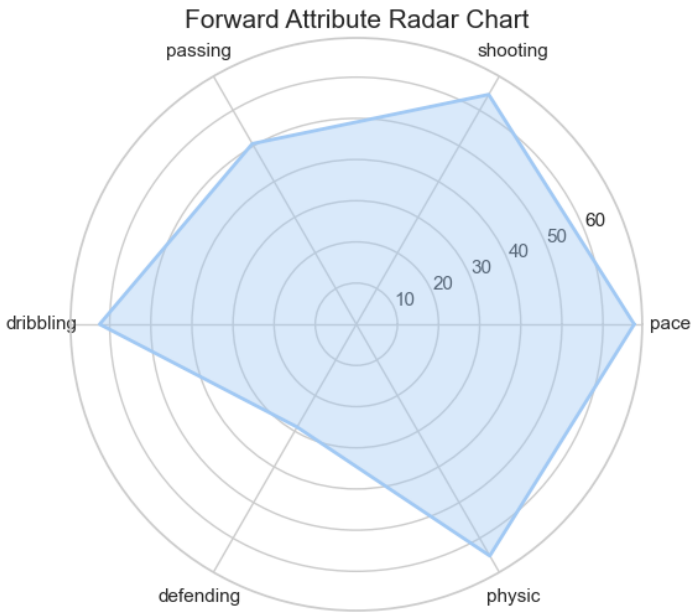
Overall, the graph confirms that wide forwards rely heavily on pace, with forwards and wing backs also benefiting from speed, while midfielders and defenders focus on other attributes in their roles.

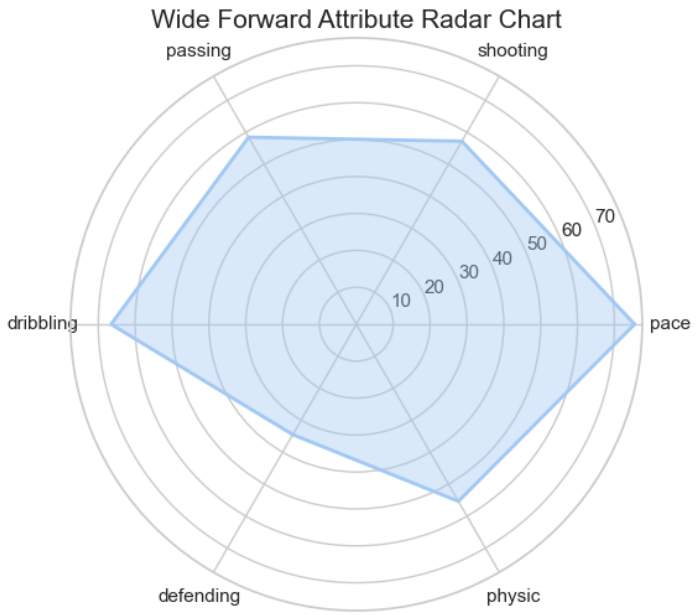


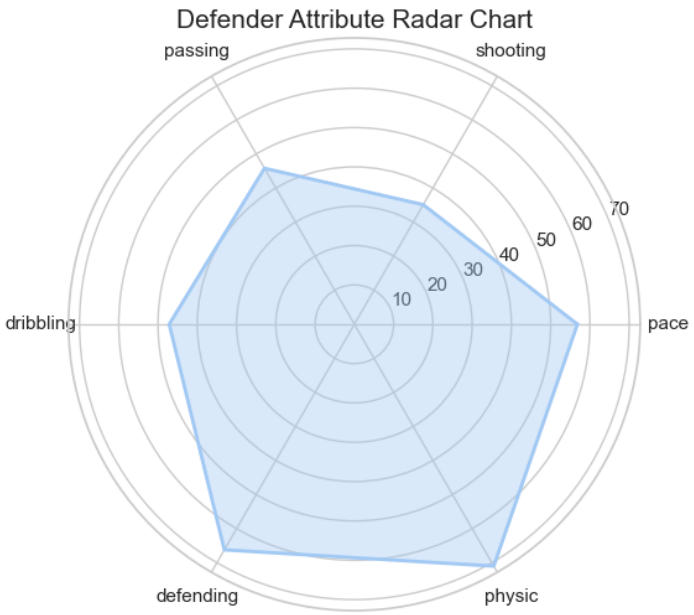
The graph above illustrates the physicality comparison across different player positions. Defenders show the highest levels of physicality, with most scoring between 65 and 75, which reflects their need to win duels and clear the ball effectively. Forwards also display high physicality, although with slightly more variation, typically between 60 and 70, as they often need to hold off defenders in attacking situations.

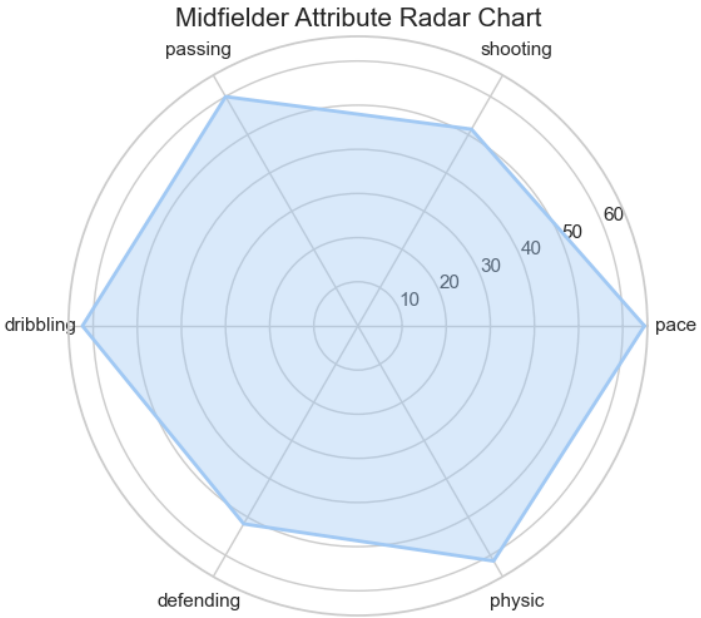
Wing backs and midfielders show more variation in physicality. Wing backs generally score between 60 and 70, reflecting their role in both attacking and defending, while midfielders have a broader range, with scores typically falling between 55 and 70. For some midfielders, physicality is key, especially in defensive roles, while others may focus on other attributes like passing or dribbling.

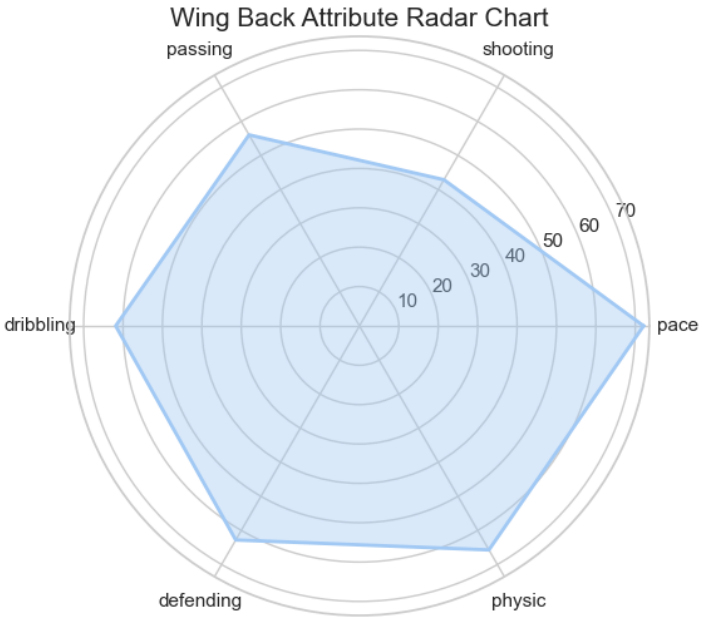
Wide forwards tend to have the lowest physicality scores, typically between 50 and 60. This suggests that they rely more on speed and dribbling rather than physical strength to succeed in their roles. Overall, this graph emphasizes that physicality is a crucial attribute for defenders and certain attacking players.


Forward Attribute Radar Chart
passing
shooting
pace
physic
defending
dribbling
10
20
30
40
50
60


Wide Forward Attribute Radar Chart
passing
shooting
pace
physic
defending
dribbling
10
20
30
40
50
60
70


Defender Attribute Radar Chart
passing
shooting
pace
physic
defending
dribbling
10
20
30
40
50
60
70

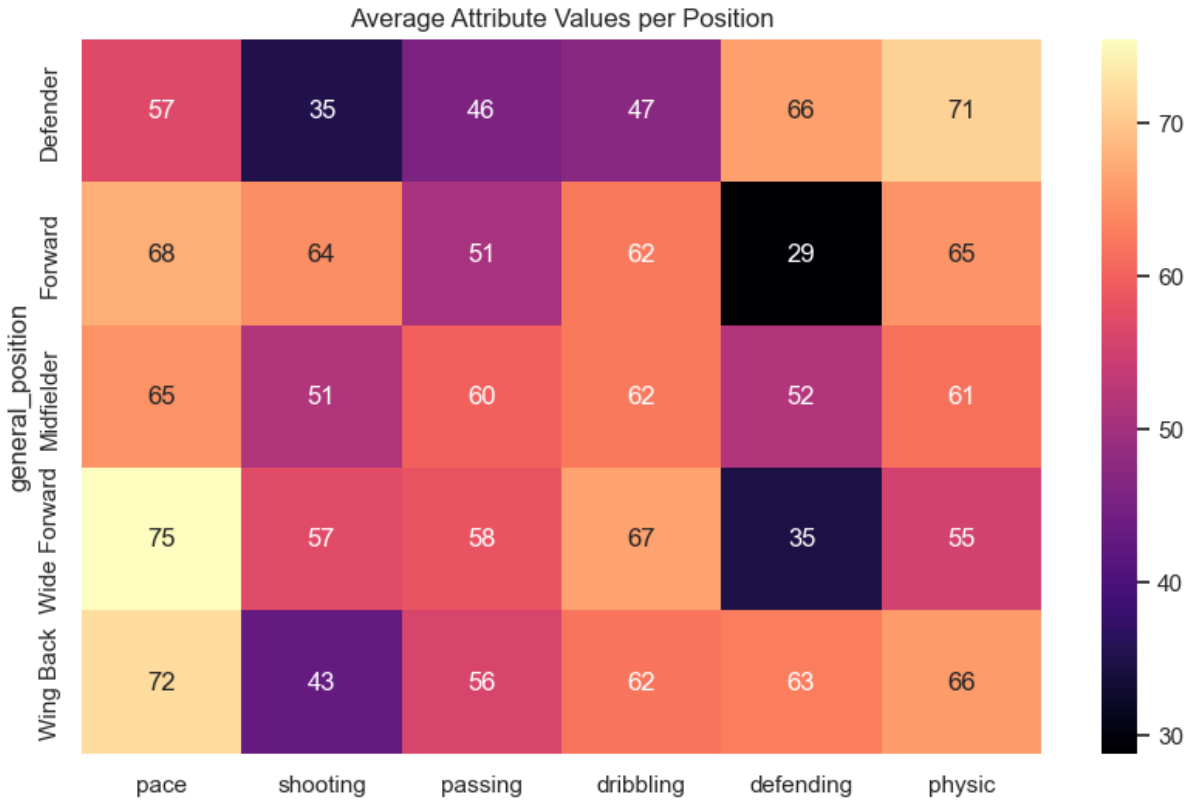

Midfielder Attribute Radar Chart
passing
shooting
pace
physic
defending
dribbling
10
20
30
40
50
60


Wing Back Attribute Radar Chart
passing
shooting
pace
physic
defending
dribbling
10
20
30
40
50
60
70

The radar charts above provide an overview of the average player attributes across each general position in the dataset. Each chart displays the relative strengths of players in different positions based on key attributes like shooting, passing, dribbling, defending, and physicality.

What's interesting is how forwards and wide forwards show higher ratings in shooting and dribbling, while defenders are strongest in defending and physicality, as expected. Midfielders demonstrate a more balanced set of attributes, reflecting their all around role on the field, and wing backs have a mix of pace and defending, showcasing their involvement in both attacking and defending duties.

The plot below confirms these findings. The heatmap shows the average attribute values per position, with color intensity indicating the strength of each attribute for each position. Forwards excel in shooting and pace, defenders dominate in defending and physicality, while wide forwards are particularly strong in pace and dribbling. Midfielders maintain an all rounded attribute profile, with higher passing skills, as expected from their role.



PROCESS

The goal of this project is to use supervised learning to classify football players into general positions based on their in game attributes. The first step in this process was to understand the problem and decide on the appropriate methods to handle it. This involved several key stages, data preprocessing, model selection, hyperparameter tuning, and evaluation of the model's performance. Each decision was made carefully, with a focus on maximizing accuracy while ensuring the models were generalizable to unseen data. The process outlined below describes the journey of finding the best model for the classification task.

DATA PREPROCESSING

The dataset used in this project comes from a collection of FIFA video game editions spanning from FIFA 15 to FIFA 21. Each edition provides detailed statistics about football players, including attributes such as pace, shooting, passing, dribbling, defending, and physicality. These attributes serve as key performance indicators within the game, influencing how a player performs on the virtual pitch. The primary goal of this project is to classify players into general positions such as Forward, Midfielder, Defender, Wing Back and Wide Forward based on their in game attributes.

Given that the dataset covers six years, therefore six games, we decided to merge all the FIFA editions into one unified dataset to ensure a more substantial data size. This approach provided us with a larger and more diverse set of player statistics, allowing the model to learn from different player performances over time. It is important to note that while some players may appear in multiple editions, their statistics and positions often change as their real world performance evolves. For example, a player may be more effective as a Forward one year but shift to a Midfielder role in a later edition due to changes in team tactics or individual skill development. Merging the datasets across years allowed us to capture these dynamic changes, making the classification task more nuanced and reflective of real world football developments. Here is a good example with a postion and stats change:

By combining these various FIFA editions, the dataset offers a rich collection of players evolving statistics, giving us a unique opportunity to train the models on how players roles and attributes may shift over time. Furthermore, the dataset includes players from different leagues, nationalities, and skill levels, which helps ensure that the models do not become too specific or narrow in their predictions. This larger dataset is better suited to training machine learning models since it provides greater variety, allowing the models to generalize better across different player types and roles.

To simplify the classification task, we grouped the players specific positions into more general categories. For instance, strikers or ST and center forwards or CF were combined under the Forward category, while central midfielders or CM, attacking midfielders or CAM, and defensive midfielders or CDM were categorized as Midfielders. This broader categorization allowed us to reduce the number of classes the model needed to distinguish, making the classification task more focused and manageable, while still capturing the key differences between general player roles. See the how we generalized the position in the table below:

ST, CF	→	Forward
CM, CAM, CDM	→	Midfielder
CB	→	Defender
LB, RB, LWB, RWB	→	Wing Back
LM, RM, LW, RW	→	Wide Forward

Before training the models, we separated the dataset into features (X) and target labels (y). The features included the players in game statistics (pace, shooting…), while the labels represented the general positions we had defined. One key aspect of our dataset was that the midfielder category included the most subpositions (such as CM, CDM, and CAM), making it the largest and broadest group. Despite this imbalance, we chose not to rebalance the number of instances in each category. This decision was based on the broader nature of the midfielder position, which inherently encompasses more roles compared to others like forwards or wing backs.

To ensure the model training process was effective, we standardized the feature values using StandardScaler. This is a common technique that ensures each feature has the same scale, which helps improve the models convergence speed and performance. After scaling, we split the data into training and testing sets, allocating 80% for training the models and reserving 20% for evaluating their final performance on unseen data. Additionally, 20% of the training set was further set aside for cross validation during hyperparameter tuning, ensuring that we could rigorously evaluate the model's generalization ability and avoid overfitting.

MODEL SELECTION

We selected a variety of classification models to experiment with, each chosen for its strengths in handling multi class classification problems. The models included in our project were, k-nearest neighbors (KNN), support vector machines (SVM), decision trees, random forests, gradient boosting and naive bayes. The selection of these models was based on their known ability to handle classification tasks effectively and their varied approaches to solving the problem, providing us with diverse methods to explore. Given that our dataset involves player statistics and classifying them into different positions, these models allow us to explore both simple and complex relationships within the data, from linear separations to highly nonlinear and feature rich classifications.

K-NEAREST NEIGHBORS (KNN)

k-nearest neighbors is a simple and intuitive classification algorithm that works by looking at the k closest data points (neighbors) to the test data and assigning the majority class among those neighbors. One of the advantages of KNN is that it makes no assumptions about the underlying data distribution. However, KNN can be computationally expensive for large datasets, and the choice of k, the distance metric, and how neighbors are weighted (uniform or distance based) can greatly affect the performance of the model. We used GridSearchCV to tune these hyperparameters to find the optimal k, distance metric (Manhattan or Euclidean), and weighting scheme. For our dataset, KNN's ability to find the majority class among similar players based on statistics is a strong asset, especially since player positions are likely determined by a combination of features that are spatially close in feature space. However, we need to balance accuracy with the computational cost of large datasets.

SUPPORT VECTOR MACHINE (SVM)

SVM is another powerful classification model that tries to find the hyperplane that best separates different classes in the feature space. SVM is particularly effective for high dimensional spaces and can be very effective when there is a clear margin of separation between classes. For SVM, the key parameters to tune are the regularization parameter (C), the choice of kernel (linear or nonlinear), and the gamma parameter. These parameters affect the model's flexibility and how well it can handle complex, nonlinear data. Through GridSearchCV, we optimized these hyperparameters and found the best SVM configuration for our task. Since player statistics such as pace, shooting, and passing might have nonlinear relationships when determining a player's general position, SVM's capacity to handle nonlinear boundaries between classes makes it highly suitable for our classification task, where the separation between midfielders and forwards, for example, may not be clearcut.

DECISION TREES

Decision trees are simple, yet powerful models that create a tree like structure where each node represents a decision based on feature values. These models are highly interpretable and are often used as a starting point for classification tasks. However, they are prone to overfitting, especially if the tree is allowed to grow too deep. For this reason, it was important to tune the maximum depth of the tree, as well as other parameters like the minimum number of samples required to split a node and the minimum number of samples allowed in a leaf node. Decision trees are easy to understand but can be limited in terms of accuracy when compared to ensemble methods. For our dataset, decision trees provide a clear

interpretability advantage, as we can easily see which player statistics lead to certain positions. This model is particularly useful for initial exploration and understanding of the feature importance. However, the risk of overfitting is something to watch out for, especially since player data might have noise.

Random Forest

Random forest is an ensemble method that builds multiple decision trees and averages their predictions to improve accuracy and reduce overfitting. By randomly selecting subsets of data and features for each tree, random forest is able to generalize better than a single decision tree. For random forest, we focused on tuning the number of trees (n_estimators), the maximum depth of each tree, and the minimum samples required to split a node and to be at a leaf. While random forests are less prone to overfitting than individual trees, the computational complexity increases with the number of trees, so we had to find a balance between performance and training time. Random forest is particularly suitable for our dataset due to its ability to capture feature interactions across subsets of player statistics. By reducing overfitting, it provides a more stable and generalized model, which is crucial when dealing with potentially noisy data or feature redundancies among players attributes.

Gradient Boosting

Gradient boosting is another ensemble method that works by building trees sequentially, with each new tree trying to correct the mistakes made by the previous ones. This iterative improvement makes gradient boosting a powerful tool for reducing bias and improving model performance. However, it is sensitive to hyperparameters like the learning rate, the number of trees, and the maximum depth of each tree. These parameters were optimized using GridSearchCV, with the goal of achieving the best balance between accuracy and model complexity. Given that the relationships between player statistics and positions might be complex and subtle, gradient boosting's ability to correct for biases and refine predictions iteratively makes it highly appropriate. This model is especially useful for handling any class imbalances or nuanced statistical differences between player positions.

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem, which assumes that the features are conditionally independent given the class label. Despite this strong assumption, Naive Bayes often performs surprisingly well in many classification tasks. For our study, we used the Gaussian Naive Bayes variant, which assumes that the data follows a Gaussian (normal) distribution. We tuned the var_smoothing parameter, which controls the smoothing of the variance to avoid overfitting on small datasets. Naive Bayes works well in this context because of its efficiency and speed, especially for a dataset of this size. While it makes strong assumptions about feature independence, the simplicity of Naive Bayes allows it to handle large scale data efficiently, and it often produces competitive results when classifying player positions based on their statistics.

JUSTIFICATION OF MODEL CHOICES

In our search for scientific papers to compare our model with, we encountered challenges in finding a suitable match. While there are numerous sources that present similar methodologies for classifying player positions in FIFA, none aligned precisely with our approach. The main difference lies in how we structured our classification, we divided players into five distinct groups and intentionally excluded the goalkeeper position, whereas most existing studies retain the goalkeeper and classify players into four broader categories forward, midfielder, defender, and goalkeeper.

This distinction made direct comparisons difficult. Including the goalkeeper often results in a significantly higher classification accuracy since goalkeepers are typically easy to identify with near perfect accuracy. Additionally, the likelihood of misclassifying a wing back as a defender is higher than misclassifying a midfielder to a defender, meaning our more **granular** classification leads to a more challenging problem. As a result, comparing our model's accuracy with those that incorporate the goalkeeper role and use broader categories would not be meaningful.

However, one valuable source we found was a blog post on Towards Data Science titled "FIFA 20 Player Clustering"[2]. While this source did not exactly mirror our approach, it provided useful insights into model selection and methodology. It inspired us to experiment with various models, such as KNN, SVM, and decision trees, while also giving us ideas on how to approach clustering and classification of FIFA players. Unlike the existing models, we chose to further explore additional machine learning models and customized our hyperparameters for each to test their performance on our unique classification structure.

JUSTIFICATION OF HYPERPARAMETERS CHOICES

In this project, each set of hyperparameters was chosen deliberately to explore a wide range of model behaviors while considering computational efficiency. Our choices were based on theoretical foundations, established best practices in machine learning, and insights drawn from relevant literature and similar classification tasks.

For the k-nearest neighbors model, we tuned three key hyperparameters, the number of neighbors (n_neighbors), the weighting scheme (weights), and the distance metric (metric). We explored n_neighbors in the range of 1 to 30 to investigate how increasing the number of neighbors affects the balance between bias and variance. A smaller number of neighbors typically makes the model more sensitive to the data (increased variance), while a larger number smooths out predictions at the risk of higher bias. The weights parameter was set to both uniform and distance to determine whether all neighbors should contribute equally to the prediction or if nearer neighbors should have more influence. The inclusion of both euclidean and manhattan[3] distance metrics allowed us to explore how different geometric assumptions affected model performance, with euclidean focusing on straight line distances and manhattan looking at grid like paths.

For the Support Vector Machine model, we tuned the regularization parameter C, the kernel type, and the gamma value. The C parameter controls the trade off between maximizing the

[2] (V. *Machine learning on FIFA 20*)

[3] (*Most popular distance metrics used in KNN and when to use them*)

margin and minimizing classification errors, with smaller values of C resulting in a smoother decision boundary. We selected C values of 0.1, 1, and 10 to explore both underfitting and overfitting. The kernel function was set to rbf and linear. The Radial Basis Function (RBF) kernel is commonly used for nonlinear problems, allowing the model to handle more complex decision boundaries. The linear kernel was included for comparison in cases where the decision boundary might be a simple linear separator. The gamma parameter was fixed to scale, which adjusts the influence of each training sample on the model, preventing extreme values from dominating the decision boundary.

For the decision tree model, several parameters were considered to control tree complexity and avoid overfitting. The criterion parameter was varied between gini and entropy, both of which are standard impurity measures for tree splitting decisions. The max_depth hyperparameter, ranging from None to 50, was introduced to manage tree depth and prevent overly complex models that could overfit the training data. We experimented with min_samples_split (2, 10, and 20) and min_samples_leaf (1, 2, 5, and 10) to ensure that splits and terminal nodes contained enough data to generalize well to new data. We also incorporated the class_weight parameter, testing both None and balanced to account for potential class imbalances within the dataset.

For random forest, the primary hyperparameters we varied were the number of trees (n_estimators), the maximum tree depth (max_depth), and the splitting criteria (min_samples_split, min_samples_leaf). Testing n_estimators values of 50, 100, and 200 allowed us to find a balance between computational cost and model accuracy. We also examined max_depth values ranging from None to 30 to determine how the depth of individual trees in the forest influenced both bias and variance. Random forest is an ensemble of decision trees, so the other parameters (like min_samples_split and min_samples_leaf) were similar to those used in the decision tree model but were tuned to see how the bagging ensemble method affects generalization.

For gradient boosting, we focused on the number of estimators (n_estimators), the learning rate, and the maximum tree depth (max_depth). The choice of n_estimators values (50 and 100) was kept relatively low due to the higher computational cost of boosting methods, and the learning rate was fixed at 0.1 to ensure that each boosting step made incremental improvements to the model. A range of max_depth values (3, 5, and 7) was tested to determine the optimal level of complexity for the decision trees used in each boosting step. The subsample parameter was included to control the fraction of data used for each boosting iteration, helping to reduce variance and improve generalization.

Finally, for Naive Bayes, the primary hyperparameter we tuned was var_smoothing, which adds a small amount of variance to each feature to prevent division by zero in cases where the variance is too small. We explored values from 1e-9 to 1e-5 to see how much smoothing was necessary to balance model stability and generalization.

Here, the table shows the hyperparameters used for each model, highlighting the different configurations tested during model training.

KNN	*param_grid = {* *'n_neighbors': range(1, 31),* *'weights': ['uniform', 'distance'],* *'metric': ['euclidean', 'manhattan']* *}*
SVM	*param_grid = {* *'C': [0.1, 1, 10],* *'kernel': ['rbf', 'linear'],* *'gamma': ['scale']* *}*
Decision tree	*param_grid = {* *'criterion': ['gini', 'entropy'],* *'max_depth': [None, 10, 20, 30, 40, 50],* *'min_samples_split': [2, 10, 20],* *'min_samples_leaf': [1, 2, 5, 10],* *'class_weight': [None, 'balanced']* *}*
Random forest	*param_grid = {* *'n_estimators': [50, 100, 200],* *'max_depth': [None, 10, 20, 30],* *'min_samples_split': [2, 10],* *'min_samples_leaf': [1, 5]* *}*
Gradiant boosting	*param_grid = {* *'n_estimators': [50, 100],* *'learning_rate': [0.1],* *'max_depth': [3, 5, 7],* *'subsample': [0.8, 1.0],* *'min_samples_split': [2, 5],* *'min_samples_leaf': [1, 3]* *}*
Naïve Bayes	*param_grid = { 'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]}*

TRAINING TIME

Training the models for our dataset took around 23 hours due to the complexity of the algorithms and the range of hyperparameters tested. Models like KNN, SVM, and random forest, especially with large datasets, tend to require significant computational time. To address this, we first conducted small test runs to estimate the time needed for each model. This helped us narrow down the hyperparameters we focused on to ensure we were testing the most promising configurations

RESULTS

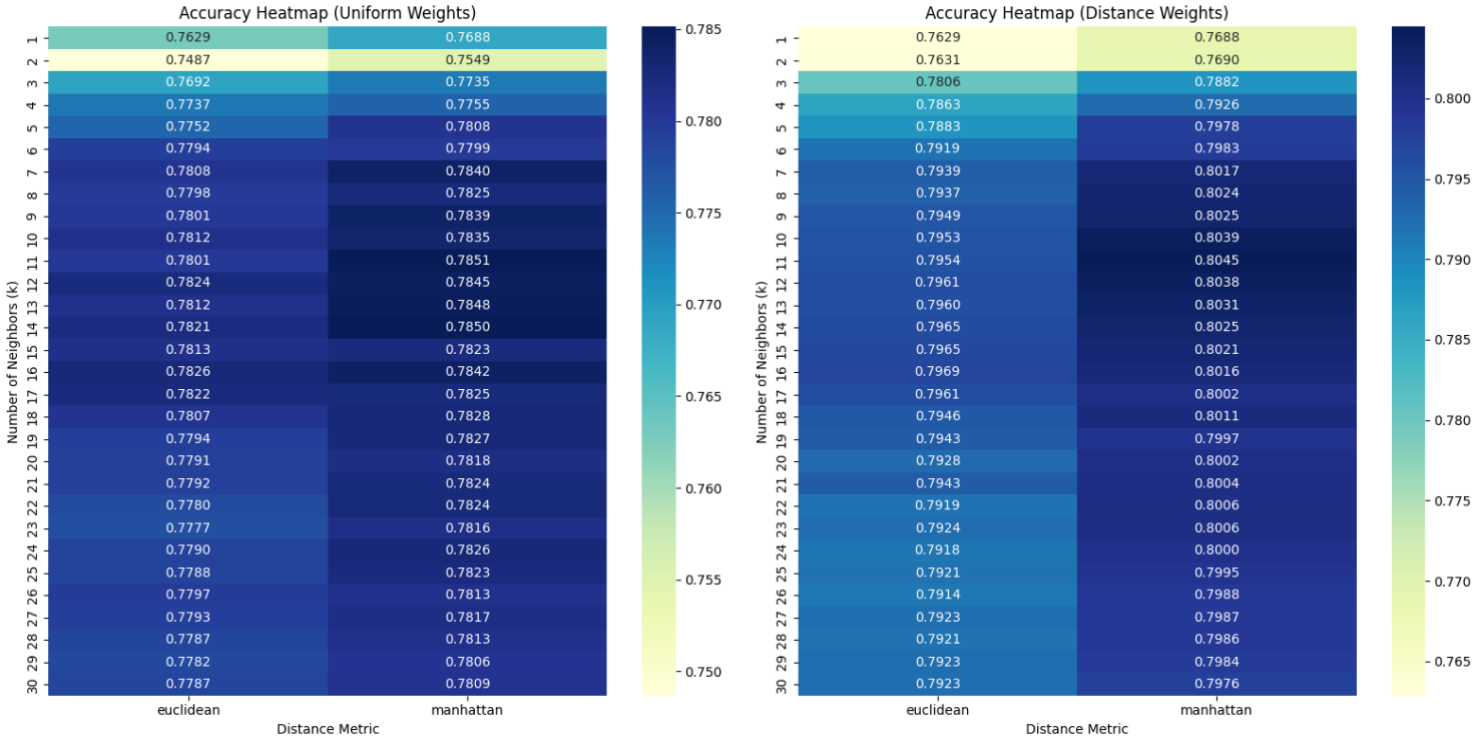
In this section, we will explore the results obtained from our machine learning models. By assessing the performance of each model and the effects of various hyperparameters, we can understand which model offers the best performance in predicting player positions based on in game statistics.

When making predictions at random, one would expect a baseline accuracy of 20% due to the nature of the multi class classification problem. Our goal is to significantly outperform this baseline by selecting and fine tuning several machine learning models.

We begin by presenting the performance of each model separately, demonstrating how different hyperparameter choices influence the model's accuracy, precision, recall, and F1-scores. For each model, we will highlight the combination of hyperparameters that produced the best results. Additionally, we will visualize these findings through tables and plots, allowing for a more intuitive comparison of model performance.

Finally, we will compare all the models and determine the best performing model based on both accuracy and computational efficiency. The insights gathered from this comparison will guide us toward selecting the most suitable model for this specific task.

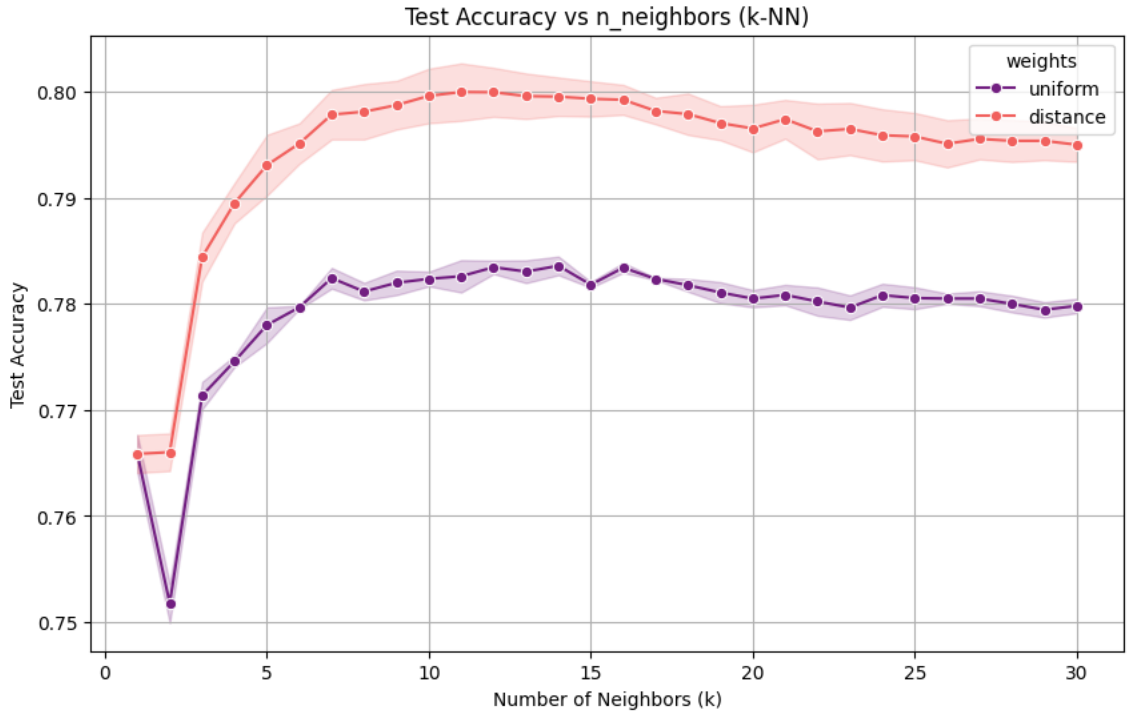
KNN



These heatmaps visualize the accuracy of the KNN model based on different combinations of distance metrics (Euclidean and Manhattan) and number of neighbors (k). The left heatmap shows the performance of the model using uniform weights, while the right one displays the accuracy using distance based weights.

In both cases, the Manhattan distance metric consistently outperforms the Euclidean metric across most values of k. This effect is particularly noticeable when the number of neighbors is small.

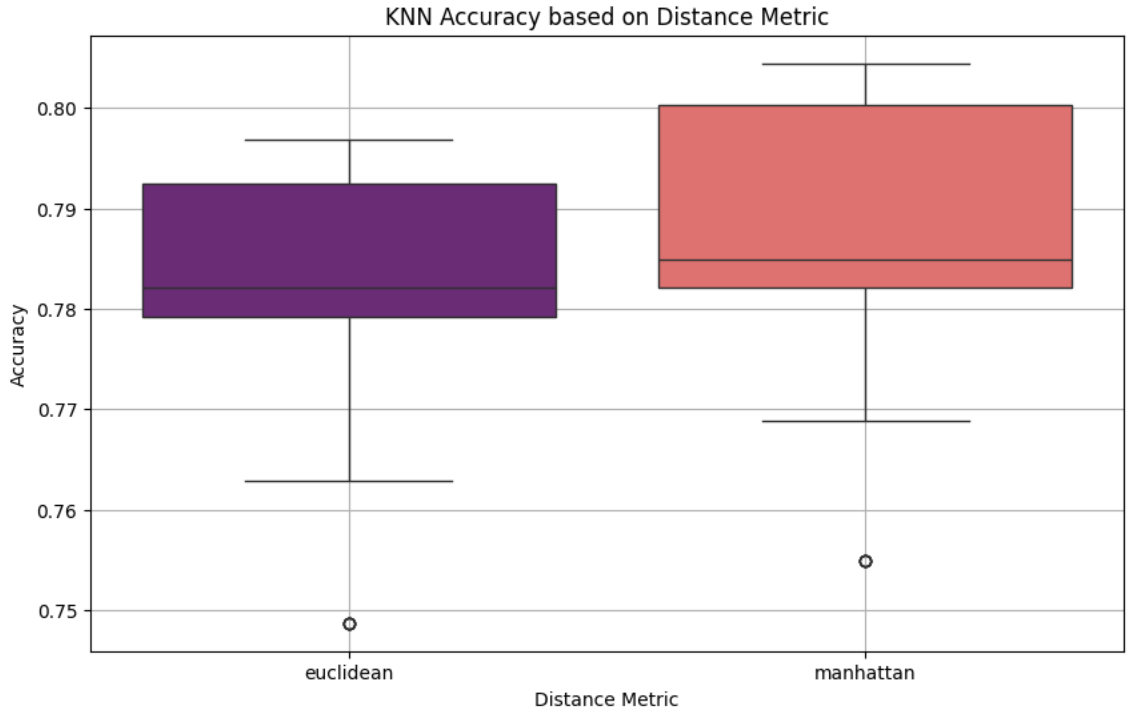
Overall, the Manhattan metric, coupled with distance based weighting, yields the highest accuracy, particularly for lower k values, where the influence of closer neighbors plays a significant role in improving model performance.



This line graph above shows how test accuracy evolves as the number of neighbors (k) increases in the KNN algorithm. Two weighting schemes are compared, "uniform" and "distance."

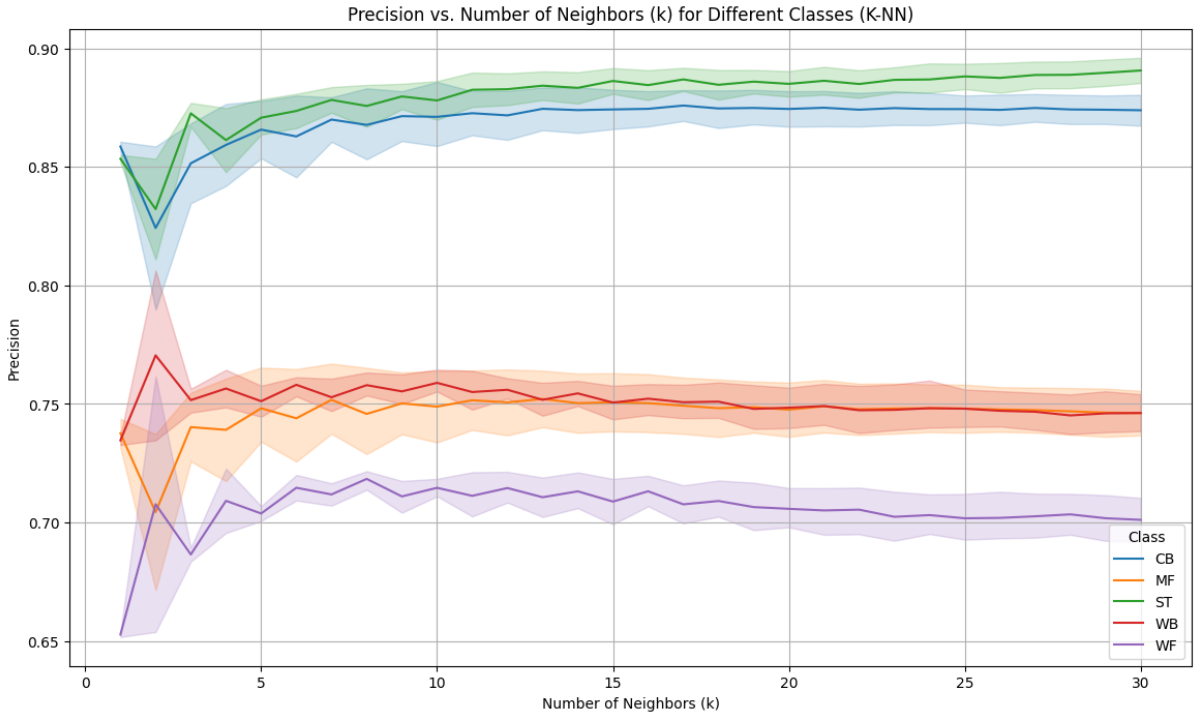
The "distance" weighting consistently outperforms "uniform" across nearly all k values, with its peak performance, occurring when k is between 7 and 18. However, as k increases, the accuracy slowly declines due to the model averaging distant neighbors. For the "uniform" weighting, the accuracy peaks around 0.784 for k 11-14, but it remains consistently lower compared to the "distance" weighting.

This graph demonstrates that the distance weighting scheme allows the model to capture more accurate information from closer neighbors, especially when k is smaller.



This box plot illustrates the distribution of accuracy for the KNN model using two distance metrics: Euclidean and Manhattan. From the graph, we can observe that the Manhattan distance metric provides higher and more consistent accuracy than Euclidean.

The Manhattan distance shows a higher median accuracy and a tighter spread, with fewer extreme outliers on the lower end. This indicates that it not only performs better in most cases but is also more consistent. The Euclidean distance, while effective in some scenarios, demonstrates more variability and occasional poor results, shown by its outliers.



This line graph shows how the precision of the KNN model varies for different player positions (CB, MF, ST, WB, WF) as the number of neighbors increases. Each class behaves differently, providing insights into how KNN performs based on position.

Strikers (ST) consistently maintain the highest precision, stabilizing around 0.88–0.89 after k reaches 10. Center-backs (CB) also show strong precision, hovering around 0.85-0.87. However, wide forwards (WF) and wing-backs (WB) exhibit lower and more fluctuating precision, particularly in the early k values, indicating that the KNN model struggles to classify these positions accurately.

Overall, this graph highlights that KNN performs better for some positions (like strikers and center-backs) than others (such as wide forwards and wing-backs), revealing positional nuances in the data.

Based on the analyses from the graphs, the best hyperparameters for the KNN model were found with the following configuration:

Distance Metric: Manhattan

Weights: Distance

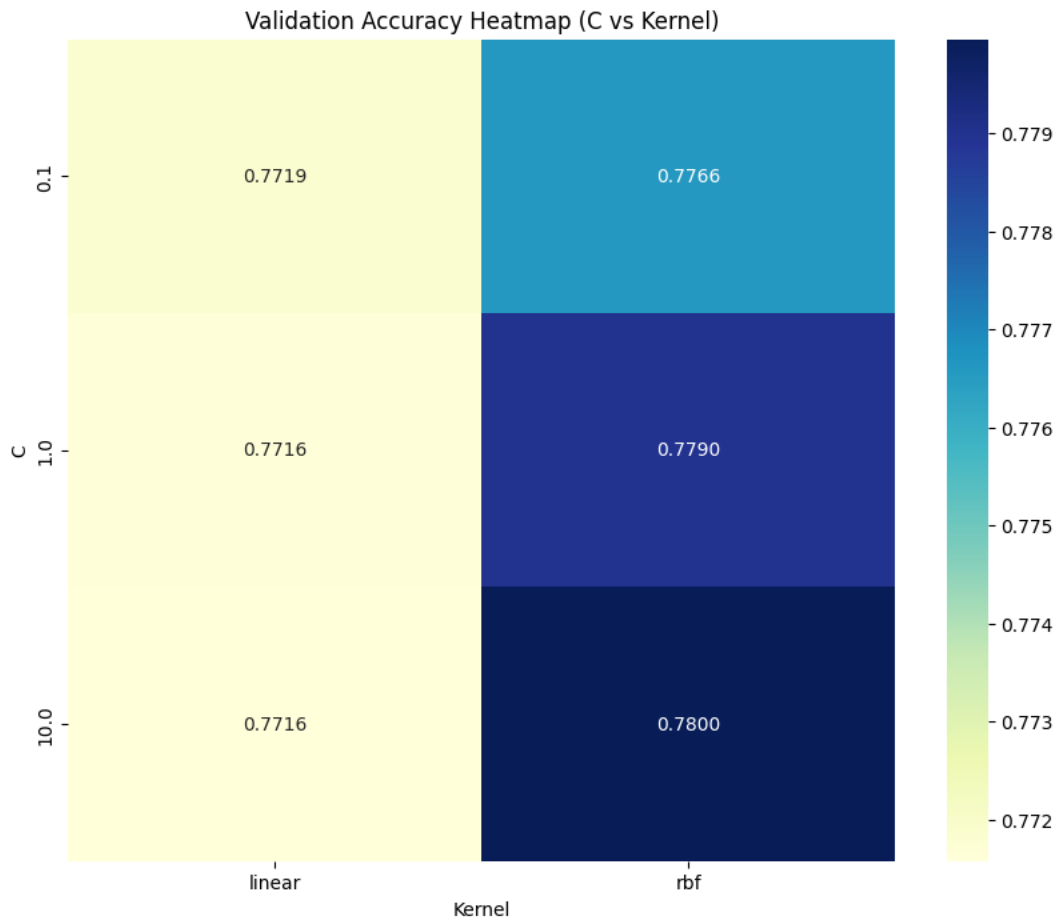
Number of Neighbors (k): 11

With these hyperparameters, the KNN model achieved the highest test accuracy of 0.8045. This configuration performed consistently better than other combinations, particularly when using higher then 7 k values, where the “distance" weighting and the Manhattan distance metric proved to be more effective in leveraging the proximity of closer neighbors to improve predictions.

The Manhattan distance metric outperformed the Euclidean metric across most tested configurations. Additionally, using distance based weights resulted in more accurate predictions compared to uniform weighting, as it gave more significance to nearer neighbors, which is crucial when working with structured datasets like this one.

The result shows that, for this dataset, choosing distance based weights and the Manhattan metric allows the KNN model to perform more effectively, especially when the number of neighbors is kept low, preventing the model from overly generalizing across distant points.

SVM



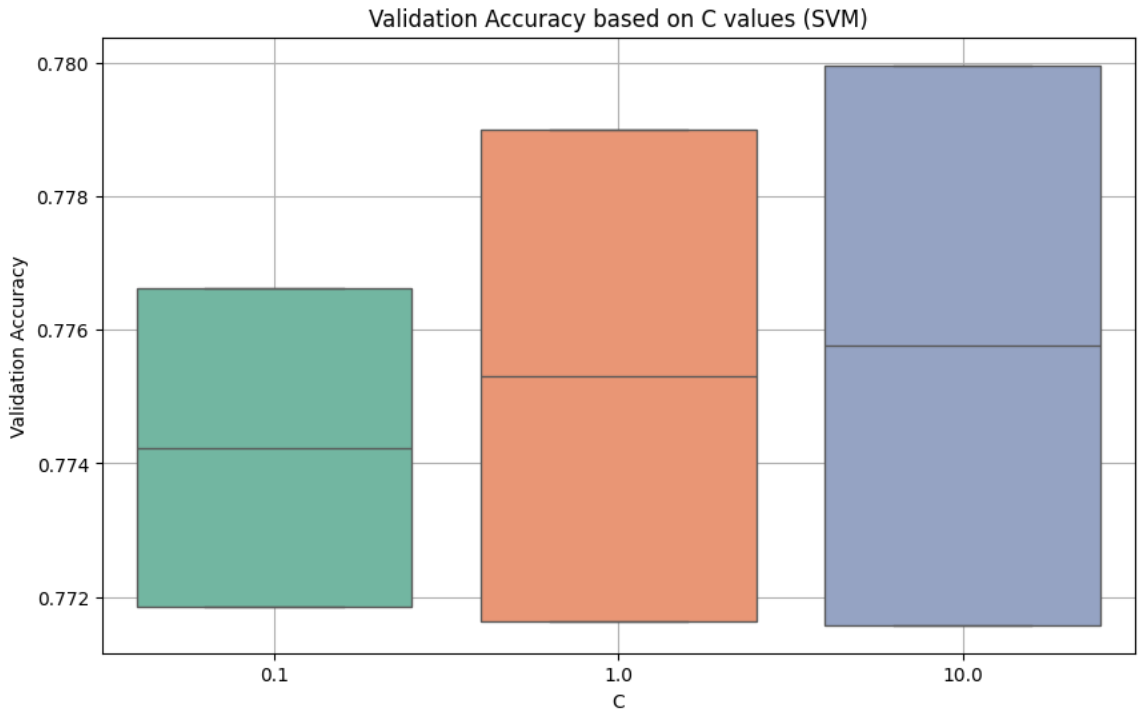
This heatmap illustrates the validation accuracy based on different values of the hyperparameter C (shown on the y-axis) and the kernel function (shown on the x-axis). The two kernel types tested are linear and rbf (Radial Basis Function), while the C values tested are 0.1, 1.0 and 10.0.

The kernel plays a significant role in determining how the data is transformed to enable better separation by the SVM. The C parameter controls the trade-off between a smooth decision boundary and correctly classifying training points. A higher C tries to fit the training data more closely.

As you can see on the heatmap the rbf kernel consistently outperforms the linear kernel for all values of C. The best validation accuracy (0.7800) is observed when using the rbf kernel with C set to 10.0. This suggests that the rbf kernel is more effective for this particular problem, allowing the model to capture more complex relationships in the data.

In contrast, the linear kernel performs relatively consistently across different C values, with a slight drop in accuracy compared to the rbf kernel. All C values tested with the linear kernel yield similar results around 0.7716-0.7719, indicating that changing C does not significantly affect performance when using this kernel.

In conclusion, the rbf kernel provides higher validation accuracy, particularly at higher C values, suggesting that this combination can generalize better for this task.

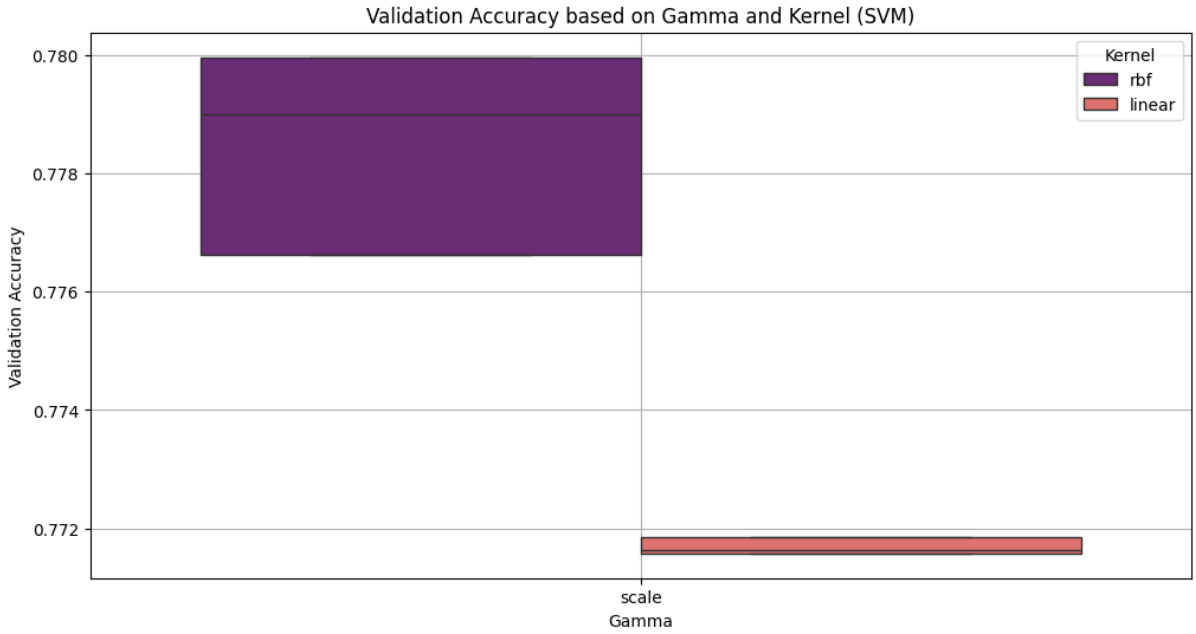


This box plot shows the validation accuracy across three different values of the regularization parameter C, 0.1, 1.0, and 10.0. The C parameter controls the trade off between maximizing the margin and minimizing classification errors. Lower values of C make the decision boundary smoother, while higher values allow the model to fit the data more closely.

From the plot, we can observe that as the value of C increases, the median validation accuracy also increases. The model achieves the highest accuracy when C is set to 10.0, with the median around 0.776. The box for C = 10.0 also has a tighter range, indicating more consistent performance compared to the lower C values.

At C = 1.0, the accuracy increases compared to C = 0.1, but it shows more variability with a slightly wider range. The median accuracy for this setting is around 0.775. Meanwhile, C = 0.1 shows the lowest accuracy, with a median just around 0.774 and a narrower distribution.

Overall, this plot suggests that increasing the value of C improves validation accuracy, with the best performance achieved at C = 10.0, likely due to the model fitting the data more effectively at higher values of C.

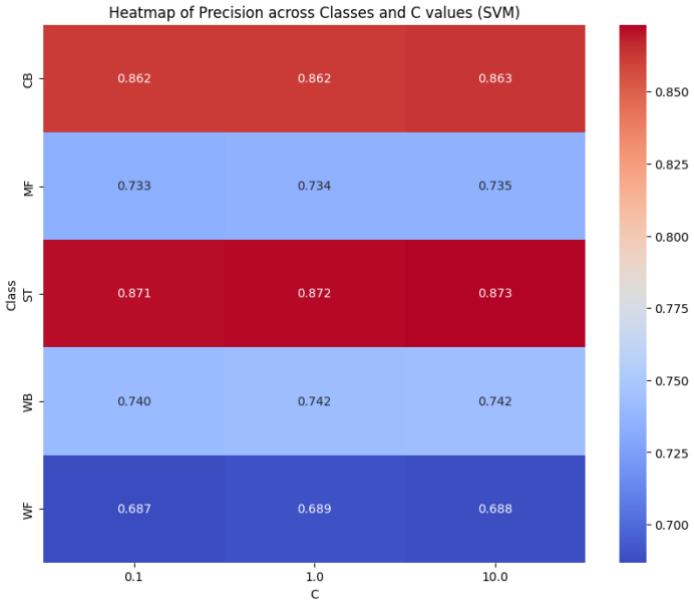


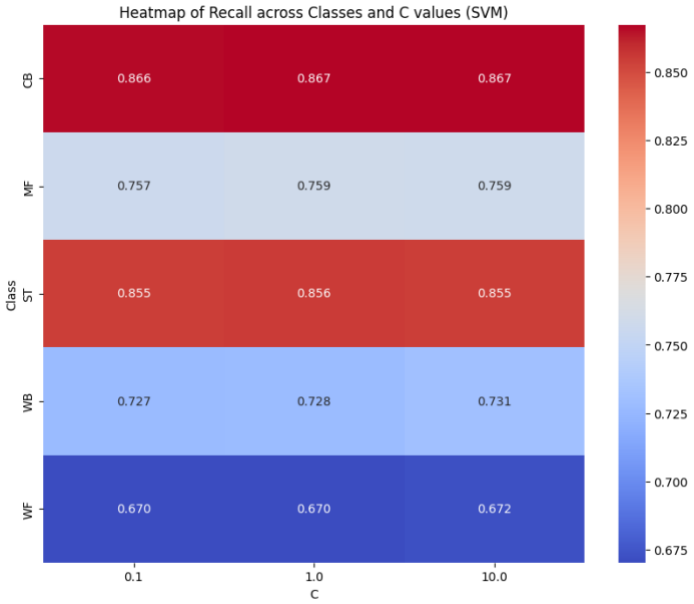
This bar plot shows the validation accuracy based on two hyperparameters, Gamma (on the x-axis) and the Kernel type. The two kernel functions being compared are the Radial Basis Function (RBF) and the Linear kernel, with Gamma set to "scale."

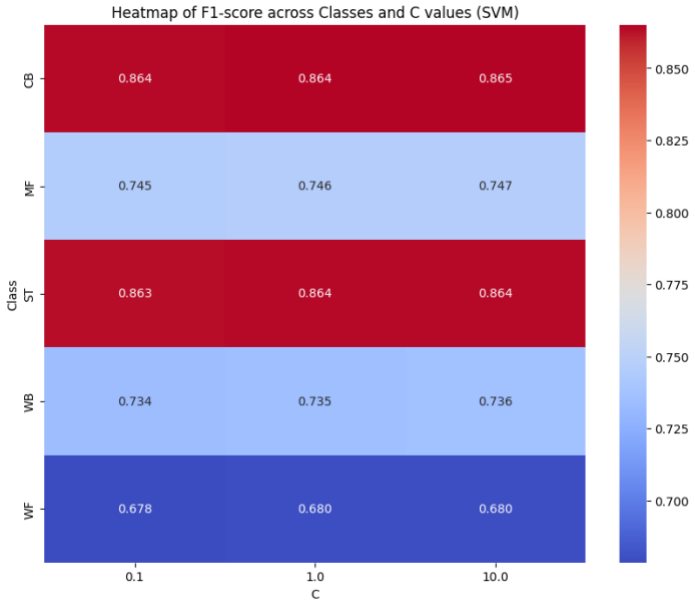
The plot clearly shows that the RBF kernel significantly outperforms the linear kernel. The RBF kernel achieves a validation accuracy close to 0.780, with a relatively small range of variability. In contrast, the linear kernel results in much lower accuracy, around 0.772, with no variability indicated in the plot.

The Gamma parameter, which controls the influence of individual training points in the model, is set to "scale," which automatically adjusts Gamma based on the number of features. It appears that the RBF kernel is able to take full advantage of this Gamma setting, capturing more complex relationships in the data and yielding much better performance compared to the linear kernel.

In summary, this plot demonstrates that the RBF kernel is much more effective than the linear kernel when Gamma is set to "scale," producing significantly higher validation accuracy for the task at hand.







These three heatmaps illustrate the performance for precision, recall, and F1-score. The y-axis represents the player positions. The x-axis shows different values of the regularization parameter C (0.1, 1.0, and 10.0).

In the precision heatmap, the Striker (ST) and Center Back (CB) positions consistently achieve the highest precision values across all C values, with precision around 0.87 for Strikers and 0.86 for Center Backs. These high values suggest that the model is particularly good at minimizing false positives when predicting these classes. Conversely, the Wing Forward (WF) position has the lowest precision, especially when C is set to 10.0, indicating a higher rate of false positives for that class. The precision for Wing Backs (WB) is moderate, hovering around 0.74 across all C values.

In the recall heatmap, similar trends are observed, with Strikers and Center Backs achieving the highest recall values, meaning the model effectively captures true positives for these positions. Notably, for the Midfielder (MF) class, the recall remains steady around 0.757-0.759, regardless of the C value, while Wing Forward again shows lower recall scores, indicating more missed true positives for this class.

The F1-score heatmap, which balances precision and recall, also shows that Strikers and Center Backs maintain high performance, with F1-scores of 0.86 for both. The Wing Forward class consistently shows the lowest F1-score across all values of C, reflecting the challenges the model faces in accurately predicting this position. The Wing Back (WB) and Midfielder (MF) positions show moderate F1-scores, slightly improving as C increases.

In summary, Strikers and Center Backs consistently perform best across all metrics, while Wing Forward struggles in terms of precision, recall, and F1-score. Adjusting the C parameter does not significantly impact the model's performance for most classes, though increasing C does seem to help slightly improve recall and F1-score for Midfielders and Wing Backs.

Based on the analysis of the SVM graphs, the best hyperparameters for the SVM model were determined with the following configuration:

Kernel: Radial Basis Function (RBF)

Regularization Parameter (C): 10.0

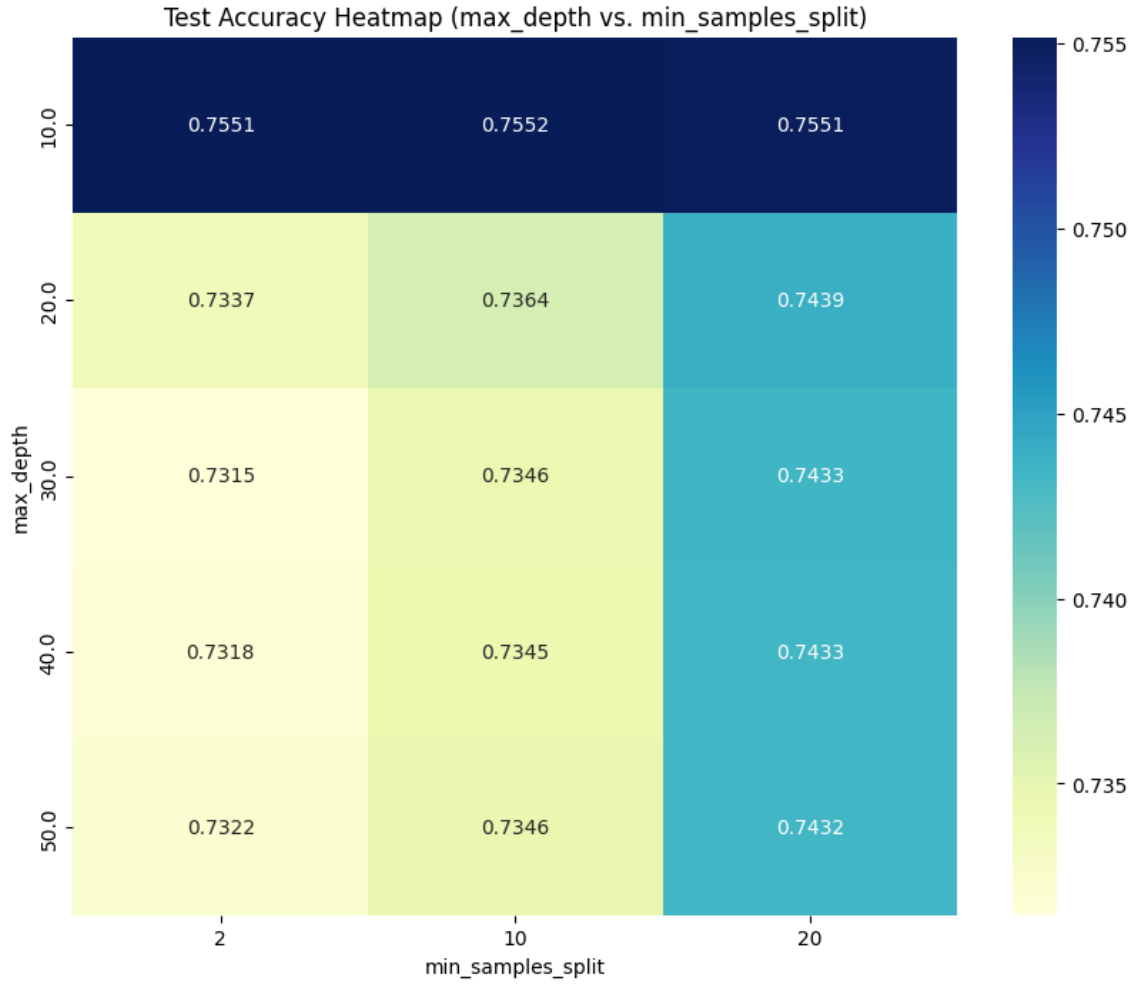
Gamma: Scale

With these hyperparameters, the SVM model achieved the highest validation accuracy of approximately 0.7800. This configuration consistently outperformed others, especially when using the RBF kernel, which allowed the model to capture more complex relationships in the dataset. The RBF kernel proved significantly more effective than the linear kernel across all values of C, demonstrating its superior ability to generalize to the underlying patterns of the data.

As the C parameter increased, the validation accuracy also improved, with C = 10.0 providing the highest accuracy and most stable performance. The C parameter controls the trade-off between maximizing the margin and minimizing classification errors, and with higher values of C, the model better fit the data, achieving more accurate predictions.

In summary, using the RBF kernel, along with a higher C value (10.0) and the default Gamma (scale), resulted in the most effective SVM model for this task. This configuration allowed the SVM to perform well in capturing the complex, non-linear relationships in the dataset, leading to the best validation performance.

DECISION TREE

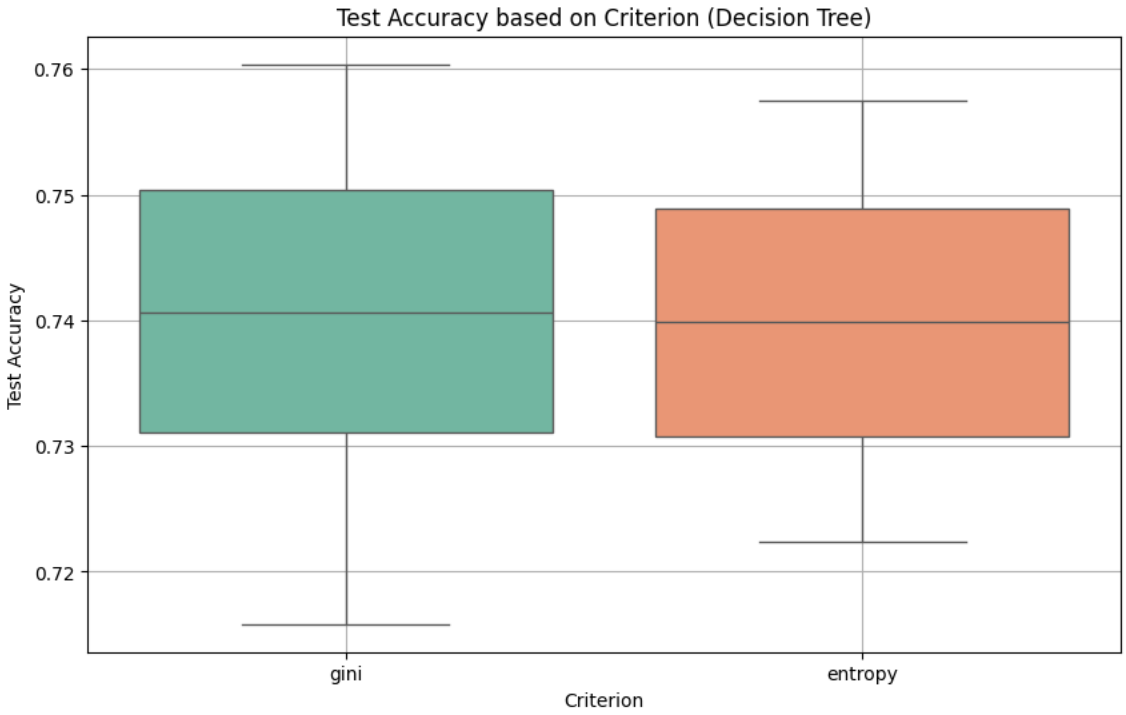


This heatmap displays the test accuracy of a decision tree model. The model's performance is measured by varying two hyperparameters, max_depth and min_samples_split.

The max_depth parameter, shown on the y-axis, controls the maximum depth of the decision tree. A smaller value restricts the tree's depth, preventing it from learning overly complex patterns. Here, max_depth values range from 10 to 50. The highest accuracy occurs at the lowest max_depth of 10, suggesting that a shallower tree might prevent overfitting and capture essential trends in player ratings without capturing noise.

The min_samples_split parameter, displayed on the x-axis, determines the minimum number of samples required to split a node, preventing splits that could create narrow branches with minimal data. The values tested here are 2, 10, and 20. The accuracy generally increases when min_samples_split is set to 20, especially at higher max_depth values, indicating that a higher sample requirement can lead to a better model fit in this context.

Overall, the best accuracy observed is approximately 0.7552, occurring at a max_depth of 10, with similar high accuracy for min_samples_split values of 2, 10, and 20 at this depth. As the depth increases, however, min_samples_split of 20 maintains relatively higher accuracy across different depths. This combination suggests that a shallow tree and a moderate to high minimum split requirement can produce the most accurate predictions for determining player positions based on FIFA ratings.



This box plot illustrates the relationship between test accuracy and the criterion used for splitting nodes in the Decision Tree model. The two criteria compared here are Gini and Entropy.

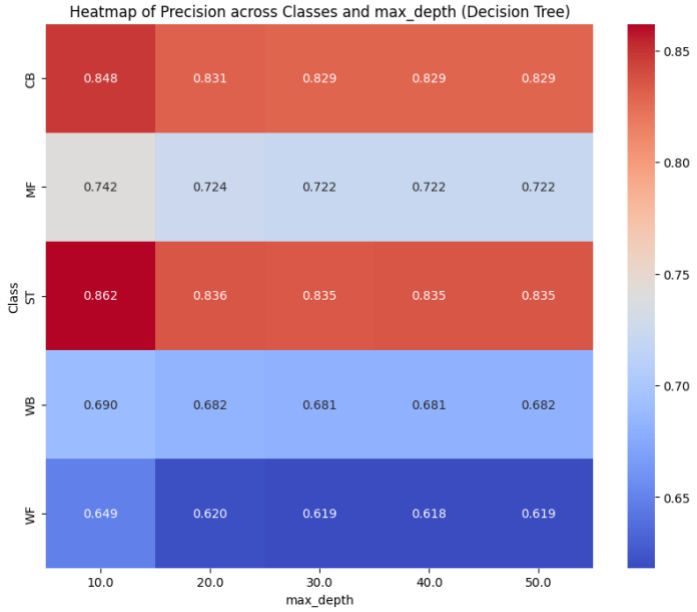
From the plot, we can observe that the Gini criterion results in a slightly higher median test accuracy compared to Entropy. The accuracy distribution for Gini shows a bit better range, with some values going over 0.75 by a bit, while Entropy has a slightly narrower range, with the majority of accuracies falling between 0.73 and 0.75. Both criteria have some overlap in their performance, but overall, Gini tends to provide slightly more consistent and slightly better accuracy outcomes than Entropy. The difference between the two, however, is not dramatic, suggesting that either criterion can be a reasonable choice depending on the specific goals of the model.

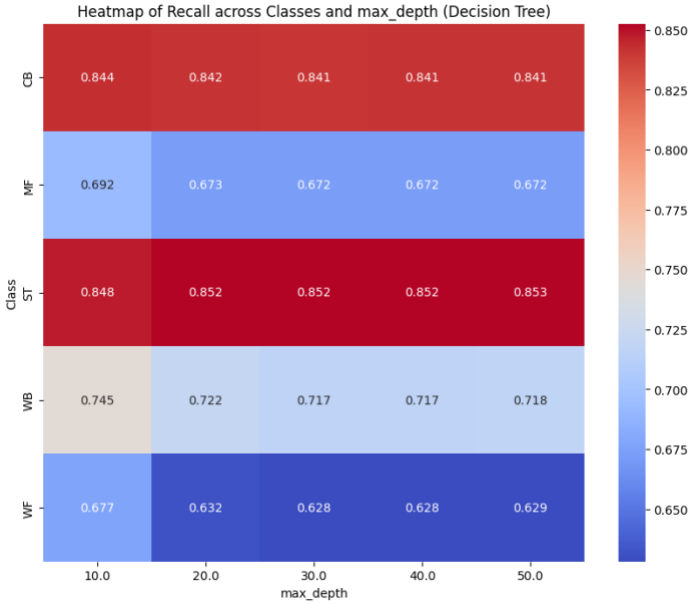
This line plot illustrates the relationship between test accuracy, the minimum number of samples required at a leaf node (min_samples_leaf), and the criterion used for splitting nodes (Gini and Entropy) in the Decision Tree model.

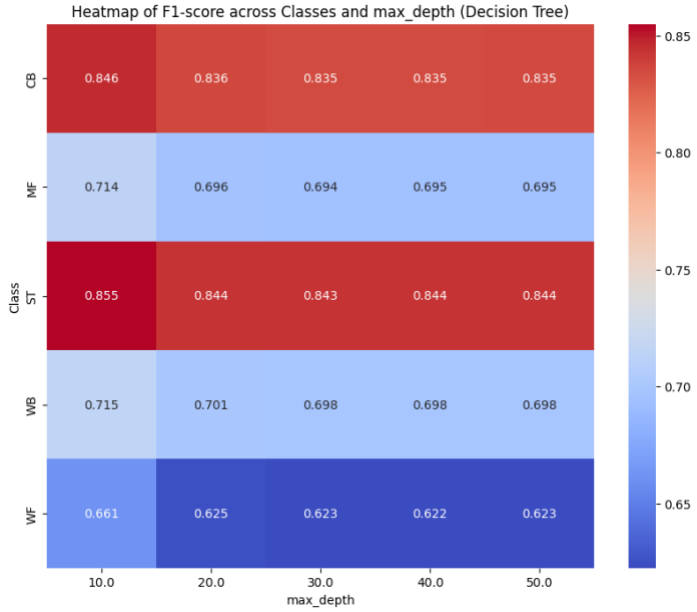
From the graph, we can observe that as the value of min_samples_leaf increases from 2 to 10, the test accuracy improves for both criteria. The Gini criterion performs better than Entropy after 4 in the min_sample_leaf. At the maximum value of 10 for min_samples_leaf, the Gini criterion reaches its highest test accuracy of approximately 0.7500, whereas Entropy lags slightly behind, peaking around 0.7475.

The increasing trend for both Gini and Entropy suggests that allowing more samples at leaf nodes helps the model generalize better by preventing overly complex trees, which might overfit the training data. However, the Gini criterion shows a sharper improvement compared to Entropy, making it the more effective choice for this dataset.

Overall, this graph indicates that the model performs best when using a higher min_samples_leaf value and the Gini criterion, providing more stability and accuracy compared to smaller leaf sizes or the Entropy criterion.







These heatmaps provide an in-depth analysis of the Decision Tree model's performance in Precision, Recall, and F1-score across various player positions. Similar to other models, the Decision Tree demonstrates strong predictive power for certain positions but faces difficulties with others.

In terms of Precision, the model performs well for CB and ST, indicating that predictions for these positions are usually accurate with fewer false positives. However, the precision for WF is noticeably lower, meaning the model tends to misclassify players as WF, resulting in more false positives for this category.

Looking at Recall, the model also excels at predicting CB and ST players, meaning it correctly identifies most actual instances in these positions. On the other hand, the recall for WF is again lower, showing that the model misses a significant number of WF players, which increases the rate of false negatives.

The F1-score, which combines both precision and recall, reaffirms these trends. CB and ST display the highest F1-scores, showing the model's balanced performance in both precision and recall for these positions. Meanwhile, WF continues to lag, with the lowest F1-scores, reflecting the model’s consistent struggles with this position across both metrics.

In conclusion, the Decision Tree model performs best at predicting CB and ST but encounters challenges with WF, where both precision and recall need improvement. Further tuning may be required to enhance the model's classification performance for WF and other positions with lower scores.

Based on the analyses from the graphs, the best hyperparameters for the Decision Tree model were found with the following configuration:

Max Depth: 10

Min Samples Split: 10

Min Samples Leaf: 10

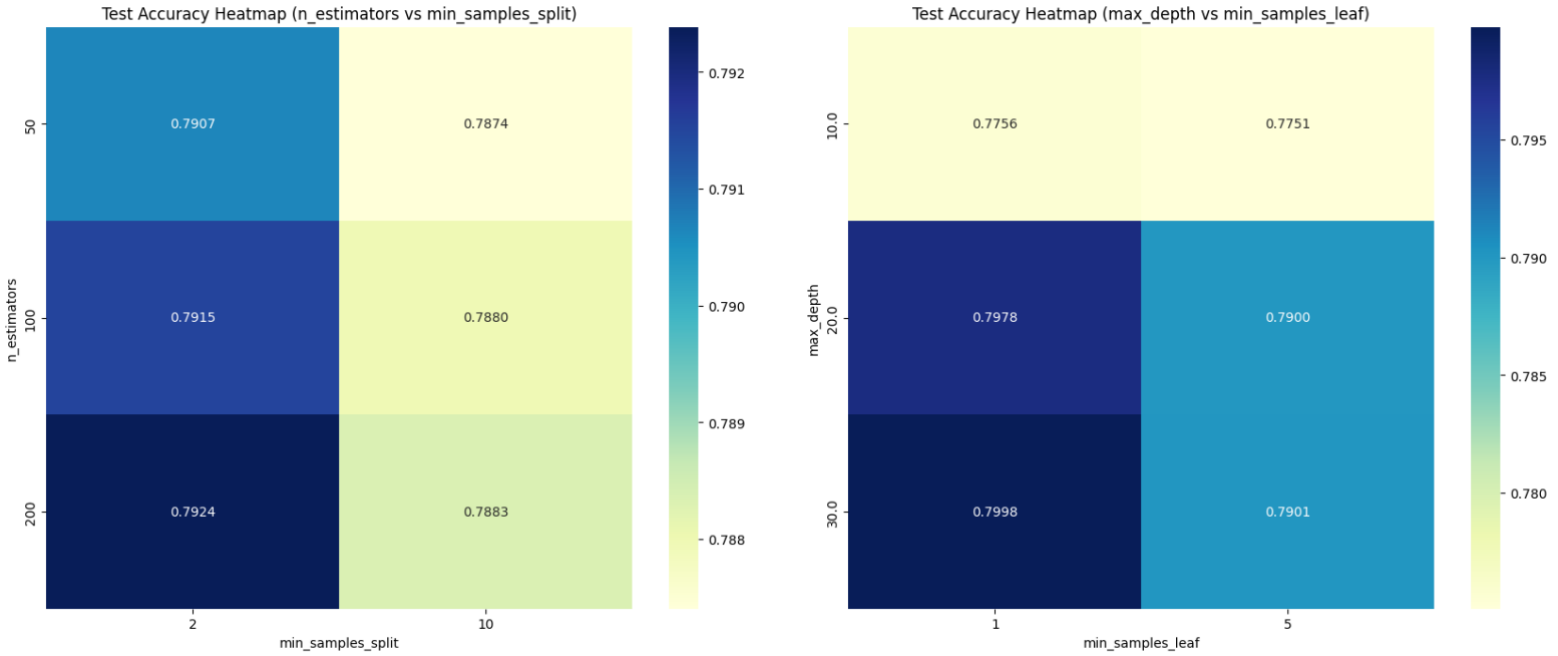
Criterion: Gini

With these hyperparameters, the decision tree model achieved the highest test accuracy of 0.7552. This configuration performed consistently better than other combinations, particularly with a max depth of 10, which prevented overfitting while capturing essential patterns in the data. The Gini criterion showed a slight advantage over Entropy in terms of accuracy, making it the preferred choice for this dataset.

In terms of splitting nodes, setting the minimum samples split to 2 allowed for a more flexible tree structure, enabling better generalization across different classes, especially when combined with a smaller max depth. The heatmaps confirmed that as max depth increased, the performance dropped slightly, suggesting that deeper trees may lead to overfitting on this dataset.

Overall, the results suggest that using a shallow tree with Gini as the splitting criterion and a low minimum samples split value allows the decision tree model to achieve its best accuracy. Further fine-tuning of these hyperparameters may be explored for even more optimized performance, particularly for challenging positions like Wide Forward (WF), where the model tends to struggle.

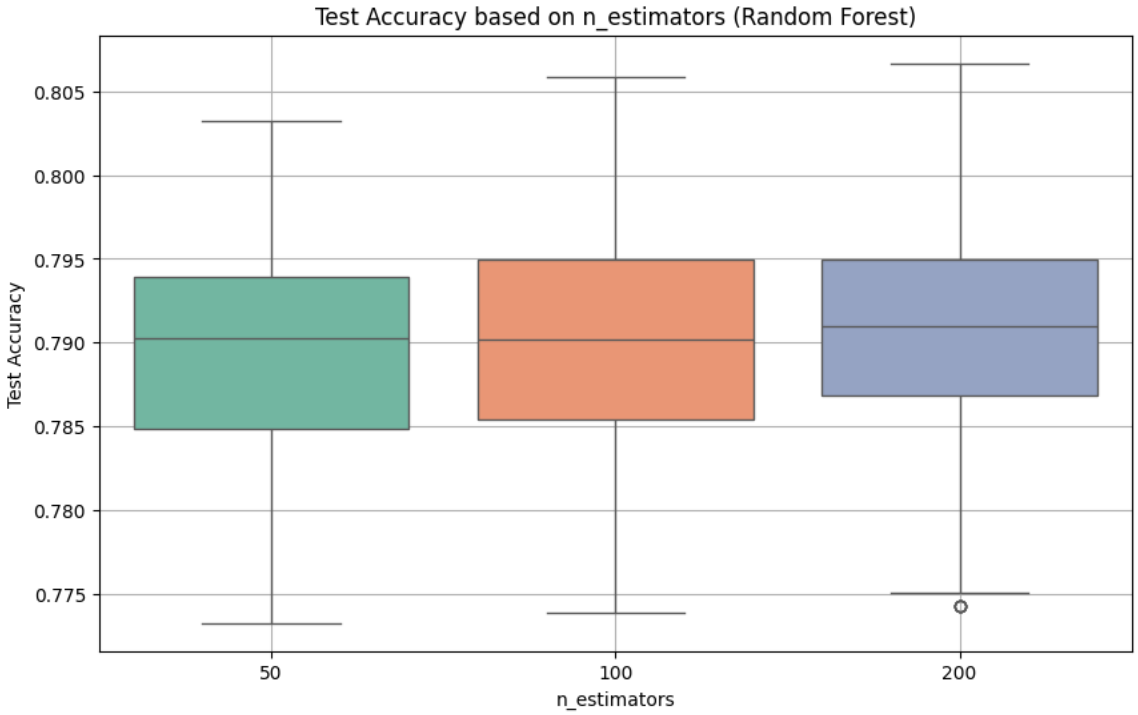
RANDOM FORREST



The heatmap on the left shows the impact of two key hyperparameters, n_estimators and min_samples_split, on the test accuracy of a Random forest model. The number of estimators (n_estimators) represents the number of trees in the forest, while min_samples_split defines the minimum number of samples required to split a node. The best test accuracy of approximately 0.7924 is achieved when the number of estimators is set to 200 and the minimum samples required to split is 2. On the other hand, reducing the number of estimators to 50 and increasing min_samples_split to 10 results in lower accuracy, around 0.7874.

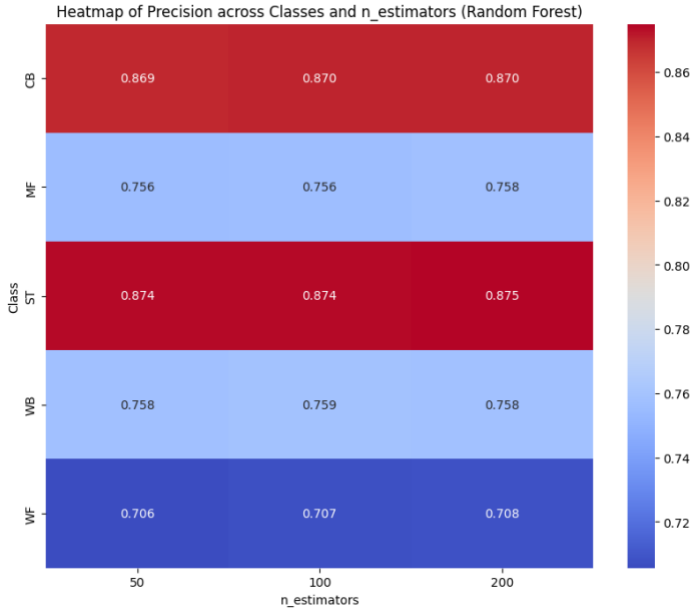
The heatmap on the right explores the relationship between the tree depth (max_depth) and the minimum number of samples per leaf (min_samples_leaf). The highest accuracy, close to 0.7998, is achieved when the max_depth is 30 and the min_samples_leaf is set to 1, indicating that allowing deeper trees with fewer samples per leaf yields better performance. In contrast, when max_depth is reduced to 10, the accuracy significantly decreases to around 0.775, regardless of the min_samples_leaf value.

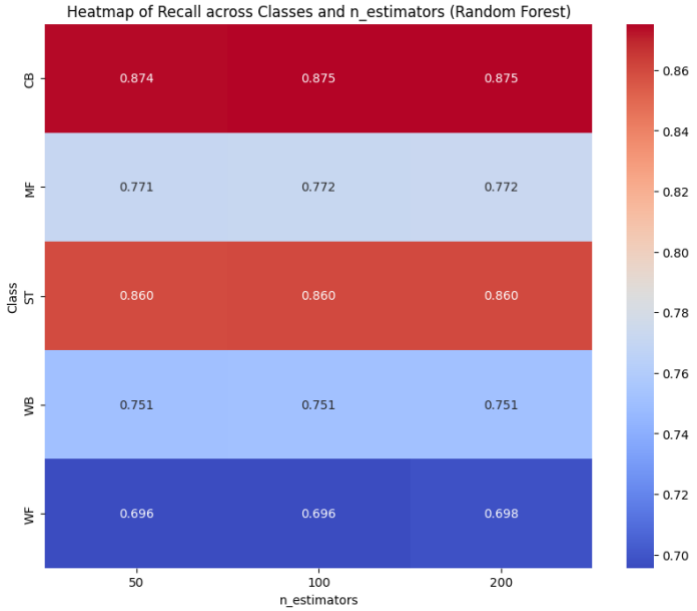
In summary, these heatmaps suggest that for optimal Random Forest performance, the model benefits from a higher number of estimators, deeper trees (increased max_depth), and smaller values for both min_samples_split and min_samples_leaf. These combinations maximize the test accuracy of the model

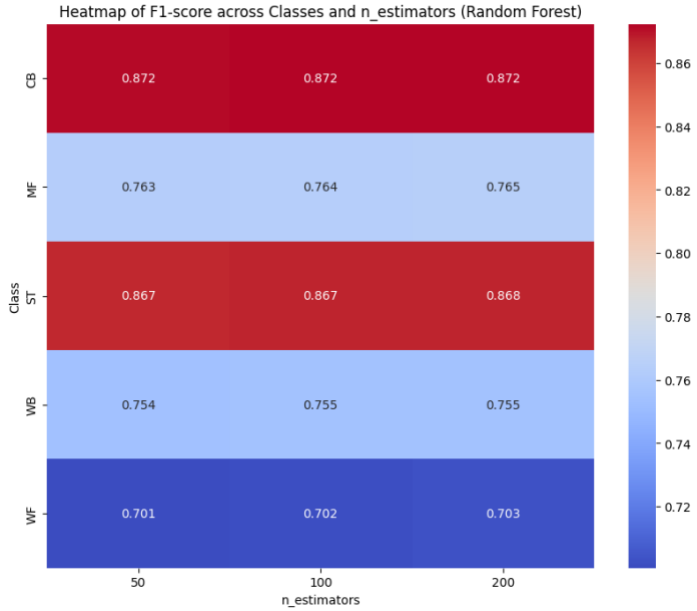


This box plot illustrates the relationship between test accuracy and the number of estimators (n_estimators) in the Random Forest model. The three different n_estimators values 50, 100, and 200 are shown, and their respective distributions of accuracy are compared.

From the plot, we can observe that as the number of estimators increases, the range of accuracies tightens, indicating more stability in the model's predictions. The highest median accuracy is observed when n_estimators is set to 200, although the difference between the median values for 50 and 100 estimators is not substantial. Additionally, the distribution for 50 estimators shows a wider range, with some outliers on the lower end, indicating a greater variance in accuracy.







These heatmaps offer a detailed view of the random forest model's performance in Precision, Recall, and F1-score across different positions.

In terms of Precision, the model achieves high accuracy for CB and ST meaning its predictions for these positions are usually correct. For WF, precision drops to 0.706, indicating a higher rate of false positives.

For Recall, CB and ST again show high scores meaning the model successfully identifies most actual players in these positions. WF, however, has a recall of 0.696, showing the model misses many actual WF players, leading to false negatives.

The F1-score, which combines precision and recall, confirms these trends. CB and ST show the highest F1-scores while WF scores the lowest, highlighting its persistent issues in both precision and recall.

In summary, the model excels at predicting CB and ST but struggles with WF, showing the need for refinement in classifying the WF position.

For the random forest model, the best hyperparameters were identified based on the analysis of the heatmaps and box plots. The optimal configuration includes:

n_estimators: 200

max_depth: 30

min_samples_split: 2

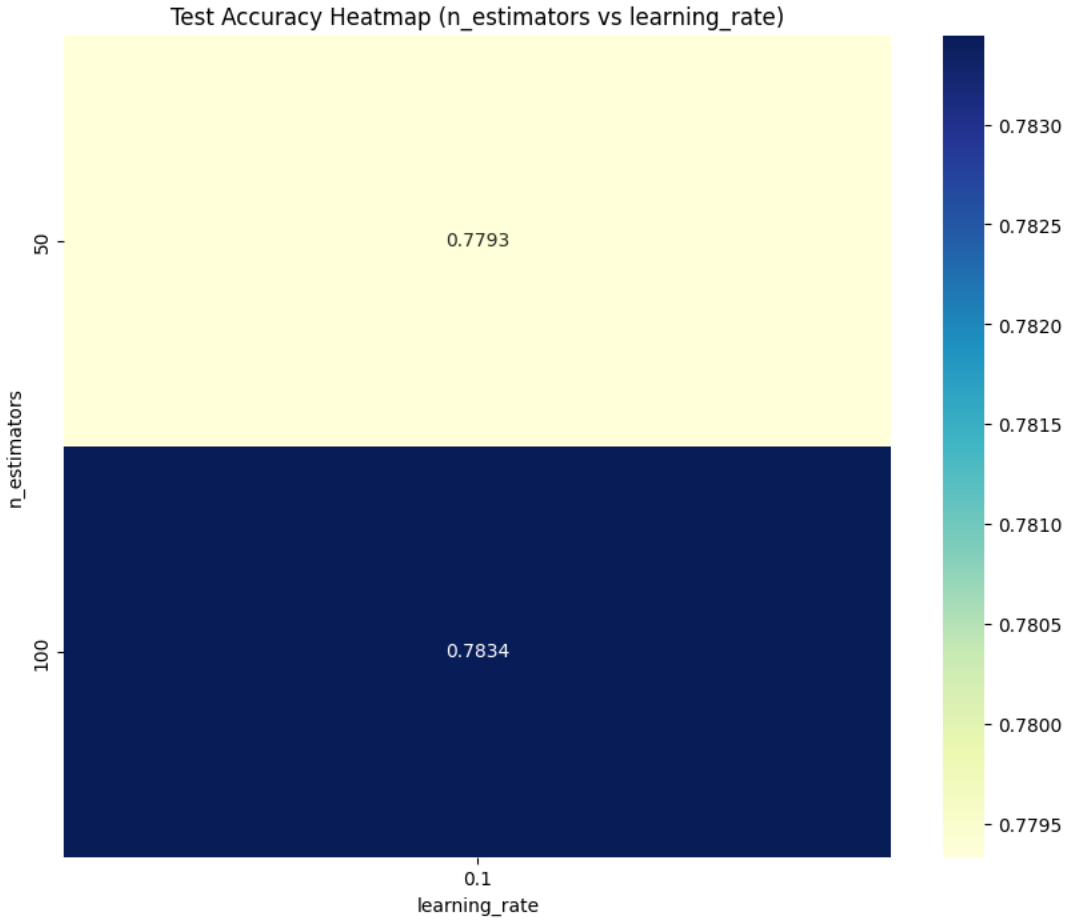
min_samples_leaf: 1

The highest test accuracy achieved with this configuration is approximately 0.7998, indicating strong performance. This setup benefits from a large number of estimators and deeper trees, allowing the model to better capture complex patterns in the data. Smaller values for min_samples_split and min_samples_leaf further improve accuracy, likely because they enable more granular splits in the data, resulting in more precise predictions.

The analysis also highlighted that increasing the number of estimators from 50 to 200 reduces the variance in accuracy, leading to more stable and consistent predictions across different runs. Deeper trees (max_depth of 30) combined with a low min_samples_leaf value (1) provide the model with enough flexibility to improve predictive accuracy without overfitting.

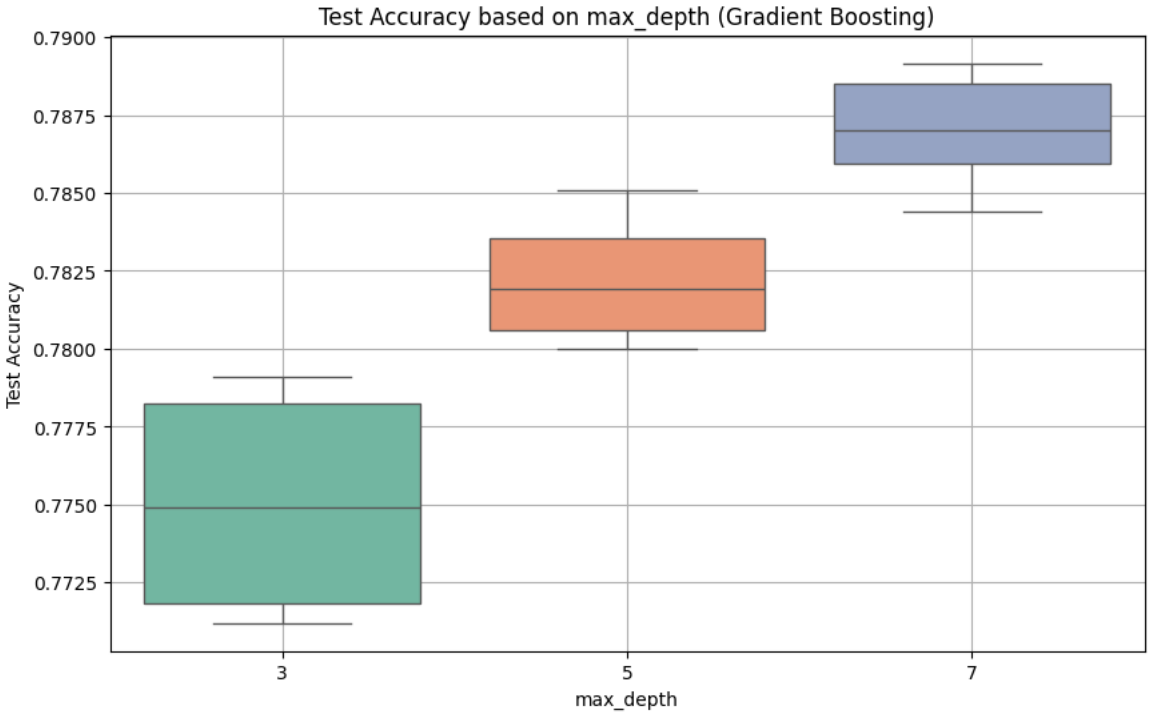
In summary, the best performing random forest configuration for this dataset leverages a high number of estimators, deep trees, and small sample sizes for splits and leaf nodes, ensuring both accuracy and stability.

GRADIANT BOOSTING



This heatmap illustrates the test accuracy based on two hyperparameters, n_estimators and learning_rate. The y-axis represents the number of boosting stages (n_estimators), while the x-axis shows the learning rate. The model's accuracy is displayed for two settings of n_estimators, 50 and 100, both with a learning rate of 0.1.

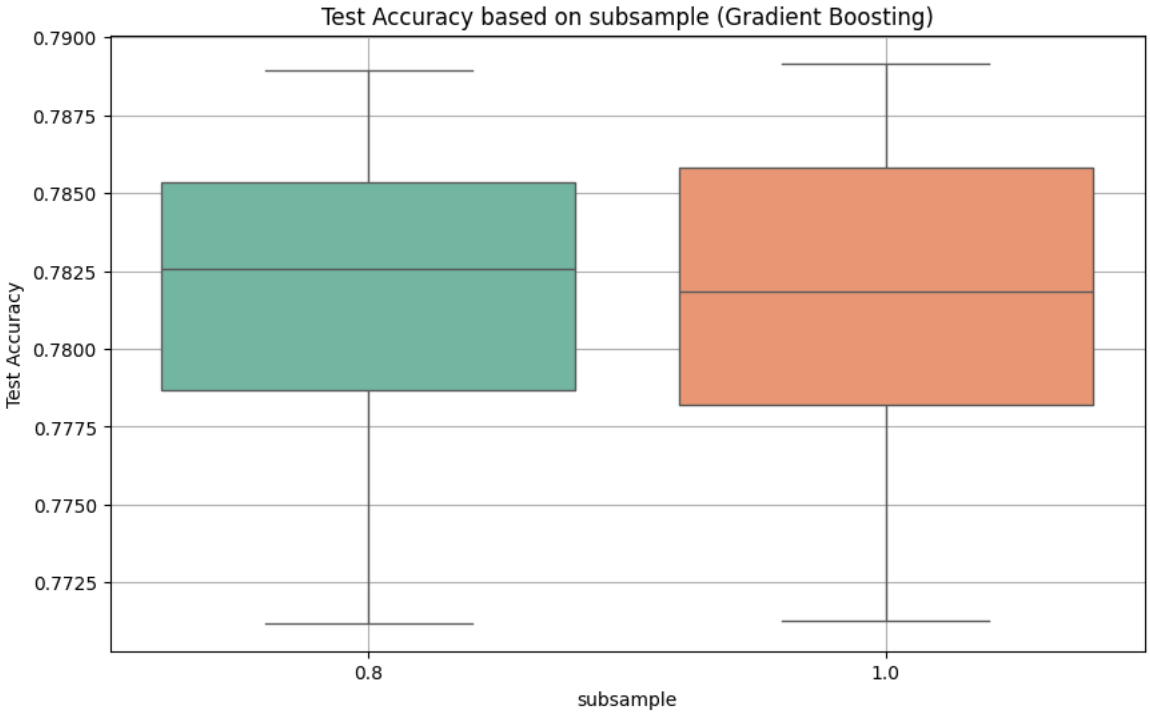
At n_estimators set to 50, the accuracy is 0.7793, while increasing the number of estimators to 100 results in a higher accuracy of 0.7834. This suggests that increasing the number of estimators improves the model's performance slightly, as the higher number of boosting stages helps the model learn more effectively, even though the learning rate remains constant at 0.1. The overall pattern indicates that adding more estimators leads to better accuracy, and a learning rate of 0.1 works well with more boosting stages to achieve better results.



This box plot shows the relationship between test accuracy and the max_depth hyperparameter. The max_depth parameter controls the maximum depth of the trees in the model, influencing how much detail the trees can capture from the data.

The plot compares test accuracy across three different max_depth values, 3, 5, and 7. The accuracy increases as the max_depth increases, with the highest accuracy observed at max_depth of 7. This suggests that deeper trees allow the model to capture more complex patterns, leading to better performance. The range of accuracy values also narrows slightly as max_depth increases, indicating more consistent performance with deeper trees. At max_depth of 3, the accuracy is the lowest, suggesting that a shallow tree may underfit the data, missing important patterns.

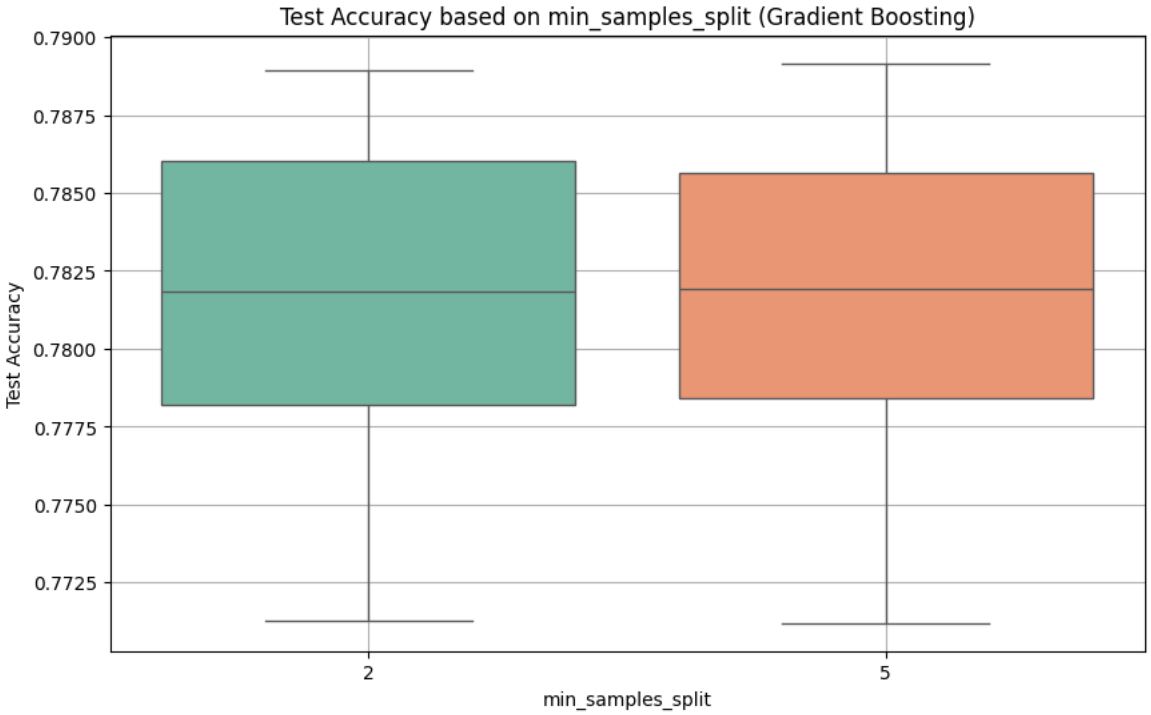
Overall, the plot suggests that increasing max_depth leads to better performance, with deeper trees achieving higher and more stable accuracy. However, the risk of overfitting may need to be considered with such deep trees.



This box plot illustrates the relationship between test accuracy and the subsample hyperparameter. The subsample parameter controls the fraction of samples used to fit each base learner, influencing how much data is used at each boosting stage.

The plot compares test accuracy for two subsample values, 0.8 and 1.0. The median test accuracy is very similar for both settings, with a slight edge for subsample of 1.0. The range of accuracy for both values is also comparable, though the model with a subsample of 1.0 shows slightly more variation in performance, as indicated by the larger spread of the whiskers.

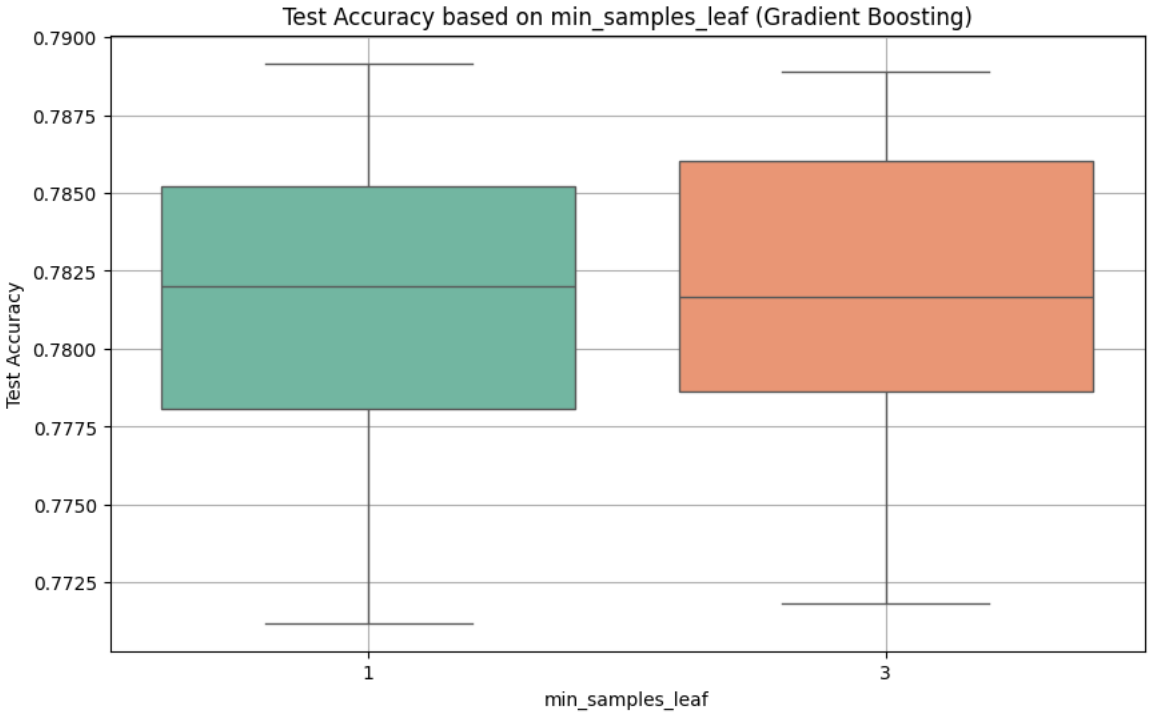
Overall, the difference in performance between the two subsample values is minimal, suggesting that using 80% or the full dataset for each base learner does not significantly affect test accuracy. However, the slight increase in accuracy at subsample of 1.0 indicates that using the full dataset might provide a marginal advantage in model performance.



This box plot illustrates the impact of the min_samples_split hyperparameter on test accuracy. The min_samples_split parameter controls the minimum number of samples required to split an internal node, influencing how flexible the model is in creating new splits.

The plot compares two values for min_samples_split, 2 and 5. The test accuracy appears relatively similar for both settings, with a slight edge for min_samples_split of 5 in terms of stability. The median accuracy for both values is close, but the range of accuracy is slightly wider for min_samples_split of 2, indicating more variability in the model's performance when fewer samples are required to split a node. In contrast, min_samples_split of 5 results in a narrower range, suggesting more consistent performance.

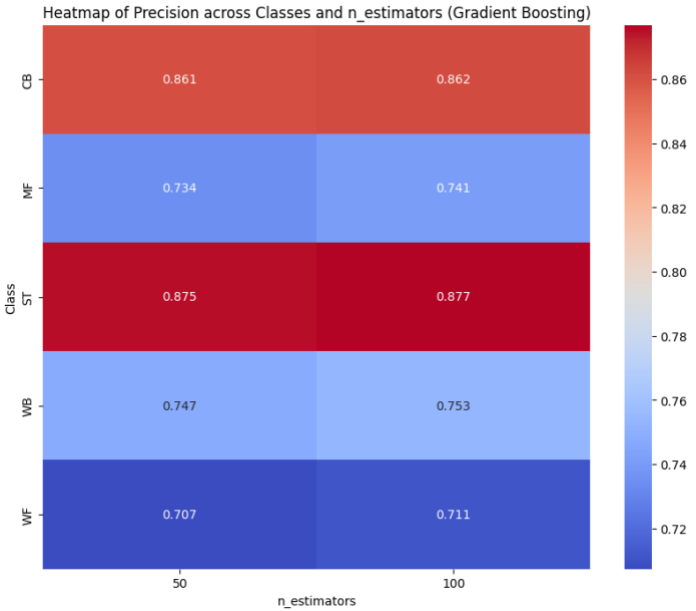
Overall, while the difference in test accuracy between the two values is minimal, the model appears to perform more reliably with a higher min_samples_split of 5, likely due to reducing the potential for overfitting by requiring more data to create splits.

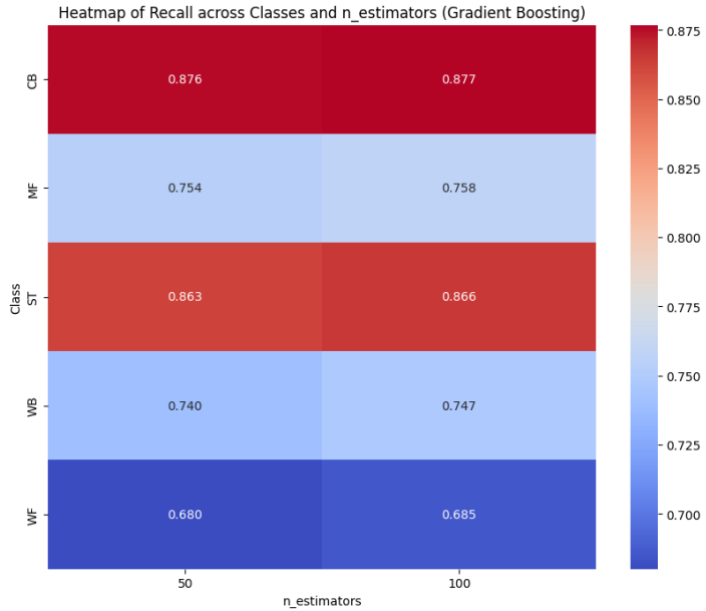


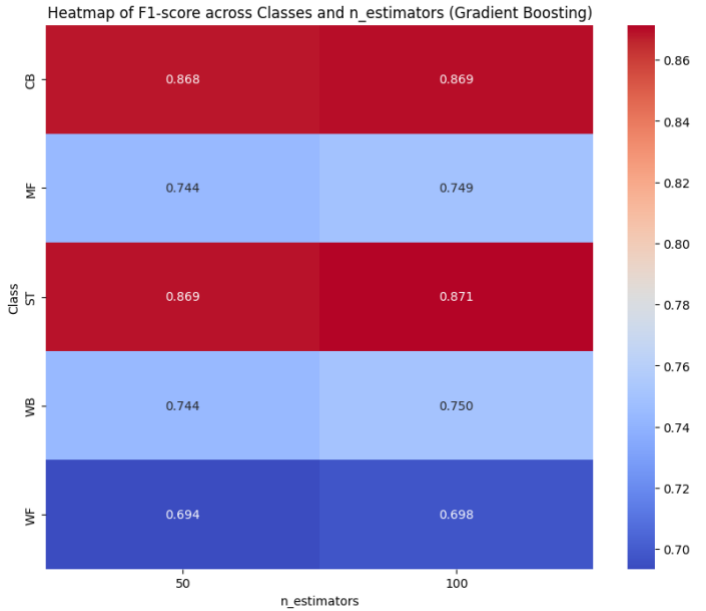
This box plot illustrates the relationship between test accuracy and the min_samples_leaf hyperparameter. The min_samples_leaf parameter sets the minimum number of samples required to be at a leaf node, which influences the size of the leaves and can help prevent overfitting.

The plot compares two values for min_samples_leaf, 1 and 3. The test accuracy is quite similar for both values, with a slight improvement in performance when min_samples_leaf is set to 3. The variability in accuracy is slightly lower for min_samples_leaf of 3, suggesting that requiring more samples per leaf results in more consistent model performance. The wider range of accuracy for min_samples_leaf of 1 may indicate that smaller leaf sizes introduce more variability in the model's ability to generalize.

Overall, while the difference in test accuracy between the two settings is minor, the model shows a slight advantage in stability and accuracy when min_samples_leaf is set to 3, likely because this setting reduces the risk of overfitting by preventing the model from creating overly small, specific leaf nodes.







These heatmaps show the performance in terms of recall, precision, and F1-score across different player positions for two values of the n_estimators parameter, which affects the model's performance. In the recall heatmap, the Center Back (CB) and Striker (ST) positions have the highest recall scores, meaning the model effectively identifies true positives for these roles. Midfielders (MF) and Wing Backs (WB) show moderate recall, improving slightly with more estimators, while the Wing Forward (WF) class has the lowest recall, indicating difficulties in correctly identifying this class.

Similarly, the precision heatmap shows strong performance for Strikers and Center Backs, with fewer false positives. Midfielders and Wing Backs have moderate precision, improving slightly with more estimators, but Wing Forwards struggle with the highest number of false positives.

The F1-score, which balances precision and recall, follows the same pattern. Center Backs and Strikers maintain the highest F1-scores, reflecting strong overall performance, while Midfielders and Wing Backs show moderate scores. Wing Forwards continue to lag behind in all three metrics.

Overall, the model performs best for Center Backs and Strikers, with consistently strong results. Midfielders and Wing Backs show moderate improvement as estimators increase, while Wing Forwards remain the most challenging. Adding more estimators slightly improves performance across most positions, but the general trend remains unchanged.

For the Gradient Boosting model, the best hyperparameters were determined from the analysis of heatmaps and box plots. The optimal configuration for this model is:

n_estimators: 100

learning_rate: 0.1

max_depth: 7

subsample: 1.0

min_samples_split: 5

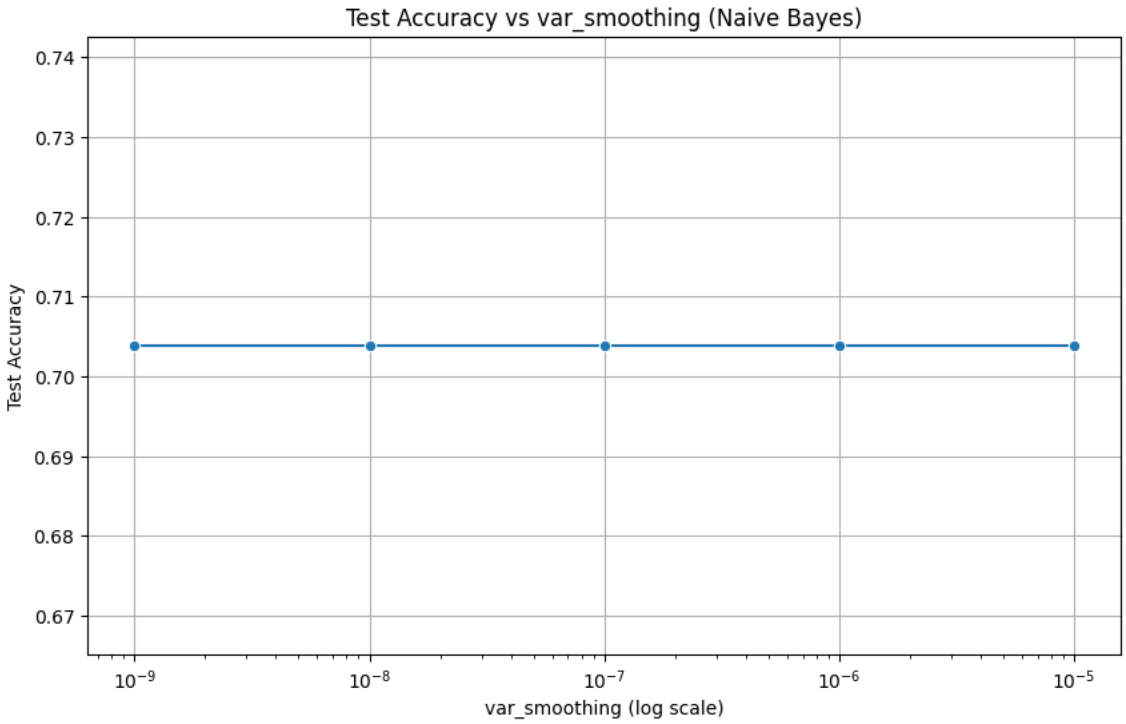
min_samples_leaf: 3

The highest test accuracy achieved with this configuration is approximately 0.7834. This setup benefits from a high number of estimators combined with a moderate learning rate, allowing the model to make incremental improvements at each boosting stage. The deeper tree structure (max_depth of 7) helps the model capture more complex patterns in the data, while using a full subsample (1.0) for each base learner slightly improves performance by utilizing all available data.

The analysis also showed that increasing the number of boosting stages (n_estimators) from 50 to 100 led to higher test accuracy, with a more stable range of performance. A higher minimum number of samples required for splitting (min_samples_split of 5) and for leaf nodes (min_samples_leaf of 3) resulted in more consistent accuracy, likely because these settings prevent overfitting by reducing the creation of overly small tree splits and leaves.

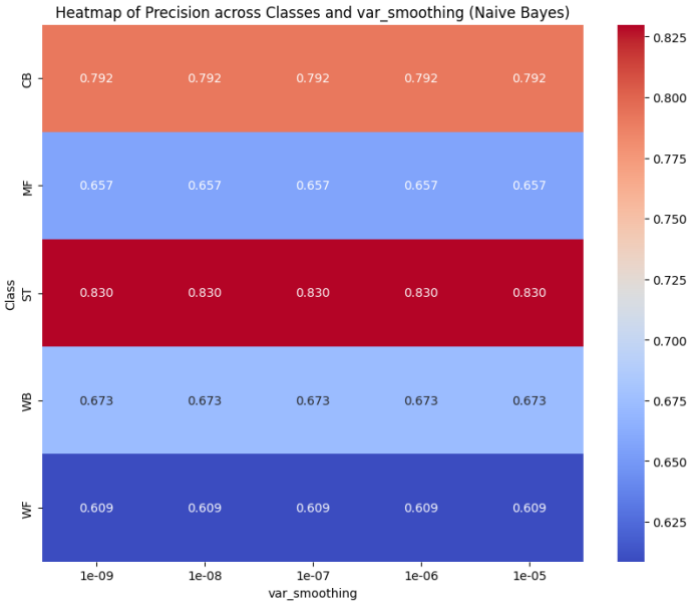
In summary, the best configuration for the Gradient Boosting model relies on a larger number of estimators, moderate tree depth, and using the full dataset in each iteration. These hyperparameter settings balance accuracy with stability, allowing the model to generalize well without overfitting to specific patterns in the training data.

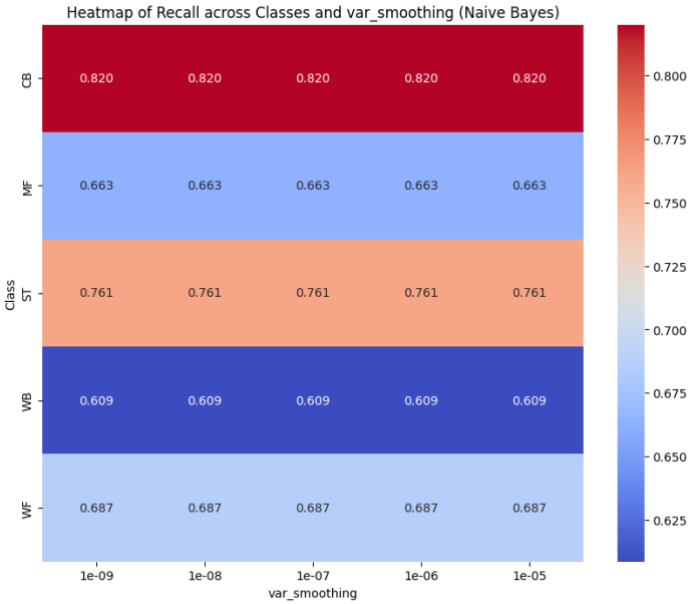
NAÏVE BAYES

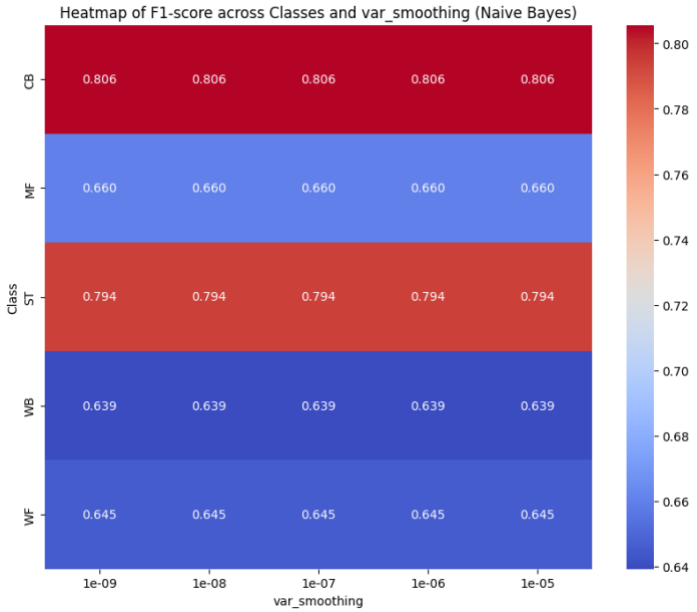


The graph above shows how the var_smoothing hyperparameter affects test accuracy, with the x-axis on a logarithmic scale. Notably, the curve remains flat, with accuracy consistently around 70%, regardless of the var_smoothing value.

This suggests that adjusting var_smoothing has no impact on the Naive Bayes model's performance for this dataset. The lack of change in accuracy highlights that the model's performance is likely limited by other factors, such as its assumptions or the dataset itself, rather than by this hyperparameter. Overall, tuning var_smoothing does not improve accuracy, indicating that Naive Bayes may not be well suited for this task.







The three heatmaps (Precision, Recall, and F1-Score) show the performance of the Naive Bayes model across different classes and varying levels of the var_smoothing hyperparameter. What is immediately noticeable across all three graphs is the consistency of the values, indicating that the var_smoothing parameter has virtually no effect on the model's performance for this dataset.

For each class (e.g., CB, ST, WF), the values for precision, recall, and F1-score remain the same regardless of the change in var_smoothing. This lack of variation suggests that Naive Bayes, in this instance, is not well suited for this dataset, as the hyperparameter tuning does not yield improvements or changes in performance.

Given the results from these graphs, it can be concluded that Naive Bayes is not a good model for this particular dataset, as it does not benefit from adjusting its hyperparameters and shows little adaptability to the dataset's complexities.

CONCLUSIONS

The results of this analysis indicate that model performance in the classification task of predicting player positions based on in-game statistics varies significantly across the different machine learning algorithms tested. The selection of appropriate hyperparameters has emerged as a critical factor in optimizing the models performance, influencing key metrics such as accuracy, precision, recall, and F1-scores. In this conclusion, we will interpret the findings of each model, highlight the importance of hyperparameter tuning, and make comparisons with other models in the literature, to the extent possible, to provide a comprehensive understanding of the outcomes.

INTERPRETATION OF RESULTS

Among the models tested, the KNN algorithm consistently outperformed the others, achieving the highest overall accuracy. With its optimal configuration (Manhattan distance, distance-based weights, and k is 11), the KNN model achieved an accuracy of 0.8045, which is significantly higher than the baseline accuracy of 20%, as well as the performance of other models. The success of KNN in this specific task can be attributed to its ability to exploit local proximity relationships between data points, especially in datasets where the distance between points has a meaningful relationship to the output variable, in this case, player positions.

A crucial finding for KNN is that Manhattan distance outperformed Euclidean distance across most configurations. This suggests that in this dataset, the feature space benefits more from measuring differences along axis-aligned paths (Manhattan) than direct straight-line distances (Euclidean), likely because of the nature of the player statistics. Additionally, distance-based weighting further enhanced the model's accuracy by emphasizing the contribution of closer neighbors. This result aligns with theoretical expectations, as closer neighbors tend to have more relevant and accurate information in KNN models. The performance of the KNN model, particularly for certain positions like strikers (ST) and center backs (CB), further highlights its suitability for this dataset, where certain positions may exhibit more distinct and localized patterns in their features.

In contrast, the SVM with a Radial Basis Function (RBF) kernel also demonstrated competitive performance, but it did not surpass KNN in overall accuracy. The SVM model achieved a maximum accuracy of 0.7800 with C=10.0 and Gamma set to "scale." The RBF kernel's ability to capture non-linear relationships in the data allowed it to perform better than the linear kernel, which struggled with the complexity of the dataset. However, SVM's performance was still inferior to KNN, possibly because SVM focuses on finding an optimal boundary between classes rather than considering local patterns like KNN does.

The decision tree model, while interpretable, did not perform as well as KNN or SVM. Its best configuration, with a max depth of 10 and minimum samples split of 10, resulted in an accuracy of 0.7552. While decision trees are powerful in capturing nonlinear relationships, they are prone to overfitting when not properly constrained. The results showed that the shallow tree with limited splits generalized better than deeper trees, which tended to overfit

the data. The relatively lower performance of the decision tree model highlights its limitations when dealing with complex, highdimensional datasets such as this one.

Similarly, the random forest model performed reasonably well, with its best configuration achieving an accuracy of 0.7998 using 200 estimators and a max depth of 30. Random forest, being an ensemble of decision trees, benefits from averaging the predictions of multiple trees, which helps mitigate overfitting compared to a single decision tree. However, despite its improved stability and accuracy, it still fell short of KNN, which leveraged its localized approach more effectively. Random orest's strength lies in its ability to model more complex interactions, but in this case, the simplicity and locality of KNN were more effective.

The Gradient Boosting model also performed well, achieving a maximum accuracy of 0.7834. The best configuration included 100 estimators, a learning rate of 0.1, a max depth of 7, and subsampling the full dataset (subsample = 1.0). The model's ability to make incremental improvements at each boosting stage, combined with a moderately deep tree structure, allowed it to capture complex patterns in the data. The configuration also helped avoid overfitting by requiring a higher number of samples for both splitting and leaf nodes, ensuring more consistent accuracy. Increasing the number of boosting stages from 50 to 100 contributed to a more stable performance range and higher accuracy overall. Although it did not outperform KNN, Gradient Boosting showed strong performance across most metrics. The Gradient Boosting model also performed well, achieving a maximum accuracy of 0.7834. The best configuration included 100 estimators, a learning rate of 0.1, a max depth of 7, and subsampling the full dataset (subsample = 1.0). The model's ability to make incremental improvements at each boosting stage, combined with a moderately deep tree structure, allowed it to capture complex patterns in the data. The configuration also helped avoid overfitting by requiring a higher number of samples for both splitting and leaf nodes, ensuring more consistent accuracy. Increasing the number of boosting stages from 50 to 100 contributed to a more stable performance range and higher accuracy overall. Although it did not outperform KNN, Gradient Boosting showed strong performance across most metrics.

The Naive Bayes model, on the other hand, underperformed significantly, with accuracy remaining around 70%, irrespective of the var_smoothing hyperparameter. This result indicates that the assumptions of Naive Bayes, mainly the independence between features were not appropriate for this dataset, which likely has more complex interactions between the features. Despite its computational efficiency, Naive Bayes was clearly not suitable for this task, as it failed to capture the complexities of the player statistics effectively.

COMPARISON WITH THE LITERATURE

Though no we found no good literature to reference, the results align with general findings in machine learning. KNN often excels in tasks where local relationships matter, and fine tuning parameters like the number of neighbors and distance metric, as seen here, is widely supported in literature. Similarly, SVM with an RBF kernel often outperforms linear models in complex tasks, but its sensitivity to hyperparameters like C and Gamma can limit its performance, as observed in this study where it couldn’t surpass KNN.

Decision trees and random forests are effective for capturing non-linear relationships, but both are prone to overfitting, especially with deeper trees, a common issue seen here. Limiting tree depth, as suggested in previous research, helped maintain reasonable accuracy.

Finally, Naive Bayes underperformed due to its assumptions of feature independence, a known limitation when dealing with more complex datasets.

IMPORTANCE OF HYPERPARAMETER TUNING

This project highlights the critical role of hyperparameter tuning in optimizing model performance. For all models tested, the choice of hyperparameters had a significant impact on the results. For KNN, the combination of distance metric, weighting scheme, and number of neighbors was essential in achieving the best performance. Similarly, the SVM model's accuracy was heavily dependent on the choice of kernel, C, and Gamma, underscoring the need for careful hyperparameter optimization in models with more complex decision boundaries.

Even for decision trees and random forests, the selection of tree depth, minimum samples split, and leaf size directly influenced the models ability to generalize from the data. For random forest, increasing the number of estimators improved stability and accuracy, while for decision trees, controlling the depth of the tree prevented overfitting. These findings reinforce the notion that hyperparameter tuning is often more critical than the model choice itself in determining overall performance.

Naive Bayes, however, showed that not all models are equally sensitive to hyperparameter tuning. Despite adjustments to var_smoothing, the model's performance did not improve, highlighting its inherent limitations rather than the impact of hyperparameter selection.

CONCLUSION

In conclusion, the KNN model emerged as the best performing model for predicting player positions based on in-game statistics. The choice of Manhattan distance and distance-based weighting for KNN played a pivotal role in maximizing accuracy, demonstrating that local proximity relationships in the dataset were key to successful classification. Although other models, such as SVM, random forest, and gradient boosting, performed competitively, their added complexity did not translate to superior performance compared to KNN. The importance of hyperparameter tuning was evident across all models, with fine tuning making the difference between mediocre and optimal performance. Overall, this project provides a strong case for using KNN in similar multiclass classification problems, particularly where the relationships between data points are meaningful in terms of proximity.

FUTURE WORK

In reviewing the results of our models, several areas for improvement become clear. While models like KNN and random forests performed reasonably well, there's room to explore methods that could yield better results, especially with complex datasets like ours.

One key takeaway is the need for a more granular look at individual categories within features. For example, instead of treating features like "pace" as a single variable, we might break it down further into its subcomponents such as acceleration and sprint speed. This would allow us to better capture the nuances in performance and interactions between these variables. Focusing on specific statistics within each category could lead to improved model accuracy by revealing patterns that are not evident when considering aggregated features.

Additionally, while we performed hyperparameter tuning, we could explore more advanced techniques to optimize our models more effectively. Instead of simple grid search, methods like randomized search or adaptive optimization might uncover better parameter combinations in less time.

A major direction for future improvement is the use of neural networks. These models are highly capable of capturing complex patterns in data that traditional models might miss. Neural networks, especially deep learning models, excel in situations where data is high dimensional and nonlinear. Though they require more computational power and fine tuning, they offer the potential for significantly better performance by learning intricate relationships in the data.

Another important area to focus on is avoiding overfitting. For models like random forests, it’s easy to overtrain on the data, resulting in models that perform well on the training set but poorly on new data. Techniques like model regularization, setting limits on tree depth, or using simpler models can help balance this.

In conclusion, while our current models provide a solid baseline, future work could focus on refining the data by using subcategories like acceleration and sprint speed, trying more advanced model optimization techniques, and introducing neural networks to improve performance. These steps would likely lead to more robust models that generalize better to new data, ultimately boosting predictive accuracy.

REFERENCES

Billcliffe, James. “FIFA 19 FUT: How Chemistry Actually Works.” *VG247*, VG247, 3 Oct. 2018, www.vg247.com/fifa-19-fut-chemistry.

Leone, Stefano. “FIFA 21 Complete Player Dataset.” *Kaggle*, 8 Oct. 2020, www.kaggle.com/datasets/stefanoleone992/fifa-21-complete-player-dataset/data.

“Most Popular Distance Metrics Used in KNN and When to Use Them.” *KDnuggets*, www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html. Accessed 1 Oct. 2024.

Oloman, Jordan. “The Best FIFA 21 Players.” *IGN*, IGN, 13 Oct. 2020, www.ign.com/articles/the-best-fifa-21-players.

V., Erick Daniel Rodriguez. “Machine Learning on FIFA 20.” *Medium*, Towards Data Science, 13 Nov. 2020, towardsdatascience.com/fifa-20-player-clustering-f500cf0792c5.