Ahmet Çeliker
2020402126

IE 582 - STATISTICAL LEARNING FOR DATA MINING

Homework 2

16.12.2024

Ahmet Çeliker - 2020402126

Ahmet Çeliker
2020402126

Introduction

This report aims to analyze sports forecasting using match odds and statistical methods, focusing on their efficacy and biases. Tasks are designed to explore bookmakers' odds accuracy, effects of specific game events, and predictive modeling for match outcomes. The analysis employs R for data handling, visualization, and modeling.

1. Task 1

1.1 Approach 1 : Earliest timestamp for each half

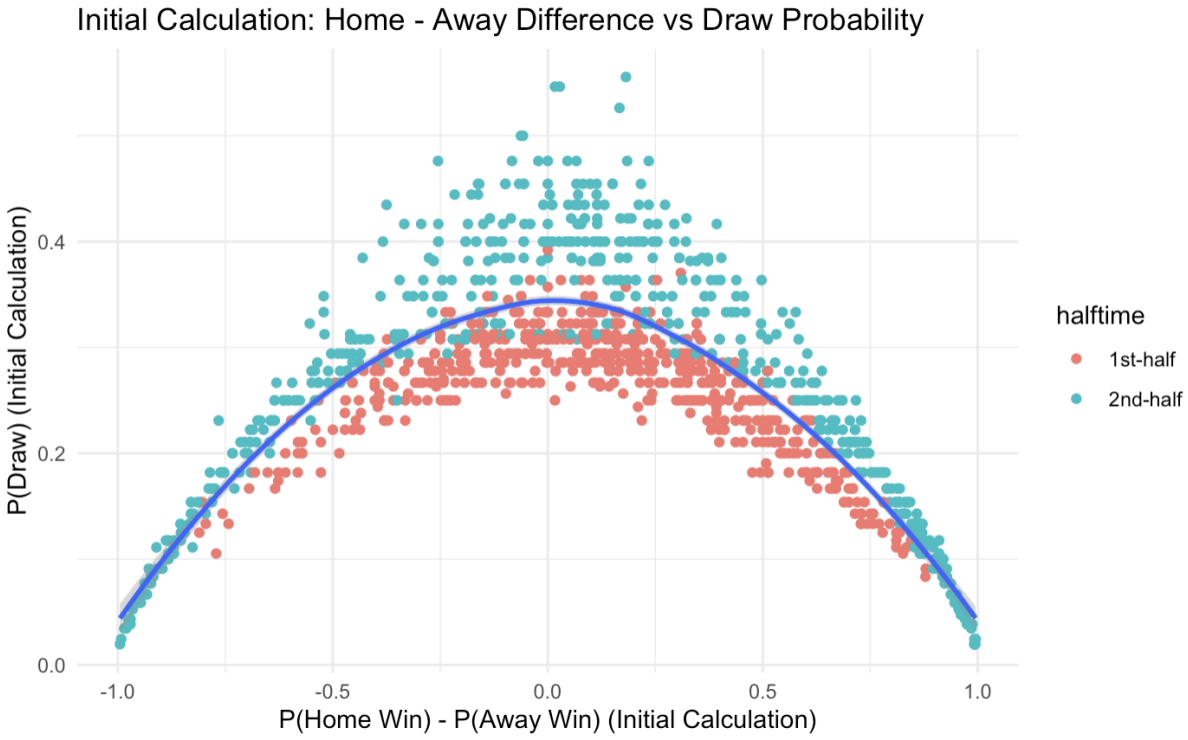
I first filtered the dataset to include only valid rows where the "suspended" and "stopped" columns were marked as "False." This ensured that my analysis focused solely on reliable data. Next, I identified the earliest recorded timestamp for each match and halftime, selecting these rows to represent the starting conditions of each game. By focusing on the earliest available data for each half, I aimed to ensure that my analysis was not influenced by biases introduced as the match progressed. Specifically, as the game nears its conclusion, the odds for the leading team tend to become significantly lower, which could create distortions in the results. By analyzing the odds probabilities from the early minutes of each half, I obtained more realistic and unbiased insights into the match conditions.

Using the betting odds provided for home win, draw, and away win (columns "X1," "X," and "X2"), I calculated the implied probabilities for each outcome using the formula. To adjust for the bookmaker's margin, I normalized these probabilities by dividing each by their total sum, resulting in adjusted probabilities for home win, draw, and away win. I then calculated the difference between the probabilities of a home win and an away win, both in their initial and normalized forms. These differences were stored as "home_away_diff" and "norm_home_away_diff," respectively, to analyze the relative strength of the two outcomes.

Ahmet Çeliker
2020402126

1.1.1 Odds and Probabilities Visualization

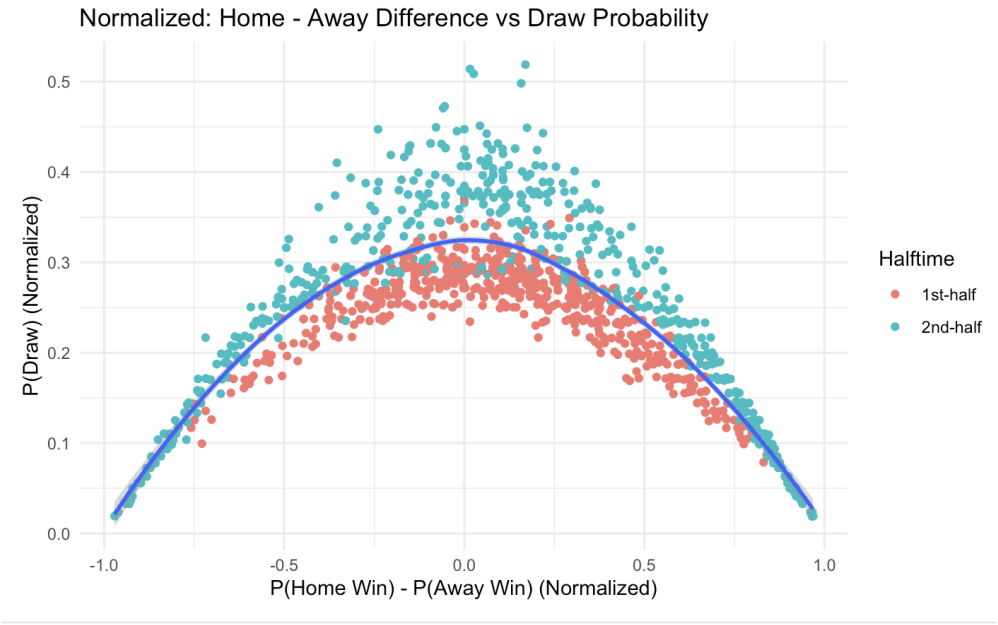
I visualized the relationship between the home-away probability difference and the probability of a draw. On the x-axis, I plotted the home-away probability difference, while the y-axis displayed the draw probability. The plot is given below.



Data points were color-coded based on the halftime (first or second), and I added a LOESS smoothing curve to capture trends in the data.

Below is the plot of the normalized data.

Ahmet Çeliker
2020402126



The key difference between the two plots lies in the normalization of probabilities. In the first plot, I used the raw probabilities calculated directly from the betting odds. These probabilities did not account for the bookmakers' margin, which often causes their total sum to exceed 1. In contrast, the second plot displays normalized probabilities, where the sum of all probabilities (home win, draw, and away win) equals 1. By applying normalization, I removed the margin introduced by bookmakers, ensuring the probabilities were mathematically consistent and comparable across different matches.

The impact of normalization was evident in the visualizations. In the second plot, the data appeared more symmetrical and consistent, particularly along the y-axis representing the draw probability. This made it easier to interpret the relationship between the home-away difference and the likelihood of a draw, providing clearer insights into the patterns present in the data.

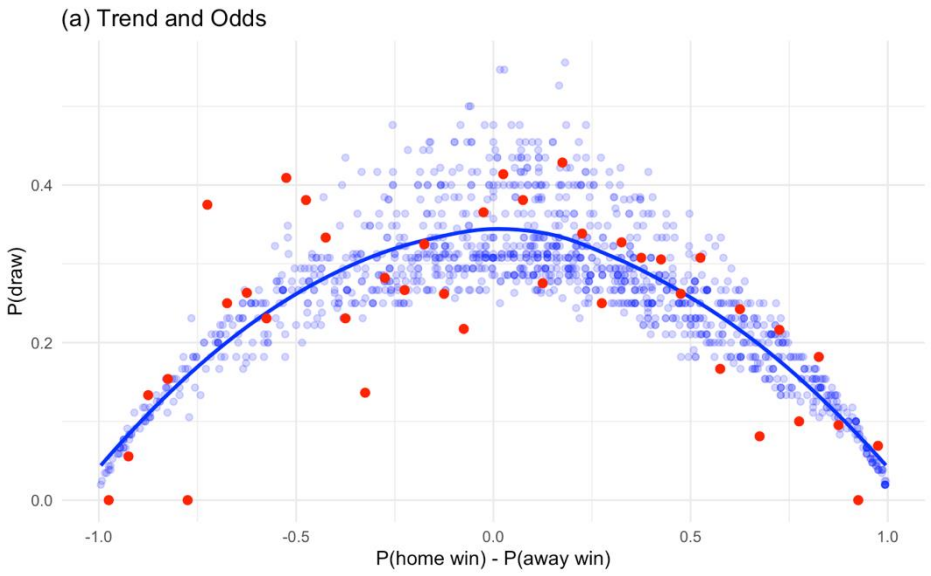
As seen above, The LOESS smoothing curve (in blue) shows a clear parabolic shape. This indicates that as the difference between the probabilities of a home win and an away win increases (moving towards either extreme of the x-axis), the probability of a draw decreases. In other words, matches where one team has a clear advantage (either home or away) are less likely to end in a draw.

The most important insight derived from the plots is that the second-half points show a sharper decline in draw probability as one team starts dominating, potentially reflecting real-time updates in market expectations.

1.1.2 Binned Data and Related Plots

I grouped the data by specific probability ranges (bins) to examine the relationship between the difference in home and away win probabilities and the likelihood of a draw. I divided the **home-away probability difference** into intervals of 0.05 using binning. For each bin, I calculated the total number of games, the number of games that ended in a draw, and the estimated draw probability as the ratio of draw games to total games in the bin.

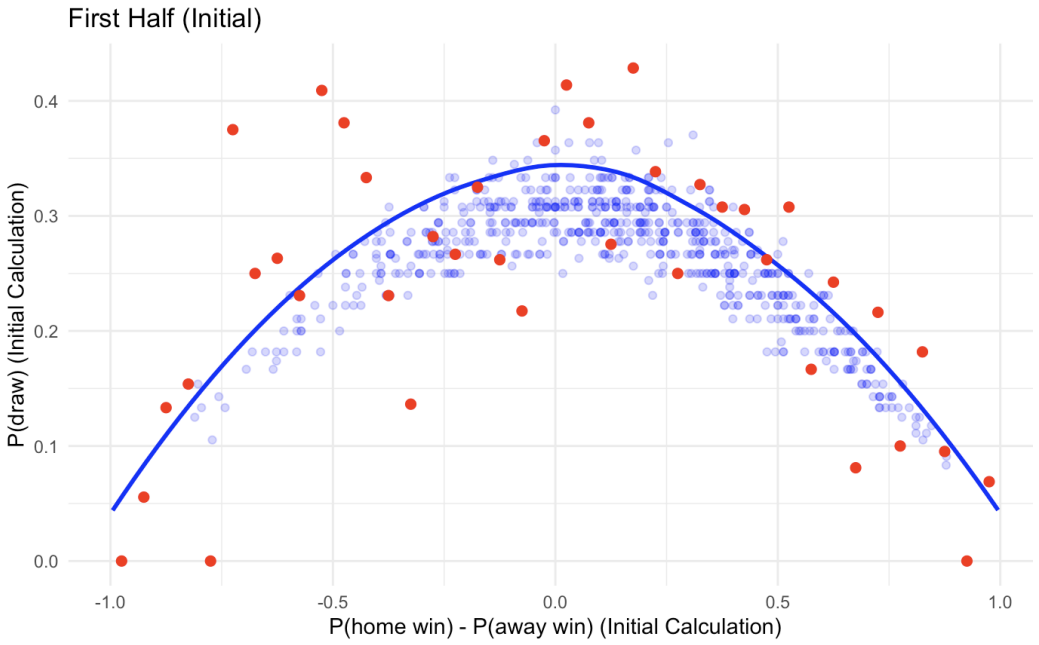
First, I plotted all the data without distinguishing between the first and second halves. The blue points represented the initial draw probabilities from the data, while a LOESS smoothing curve highlighted the general trend. I then overlaid the **binned probabilities** as red points at the bin centers to display the aggregated estimated draw probabilities.



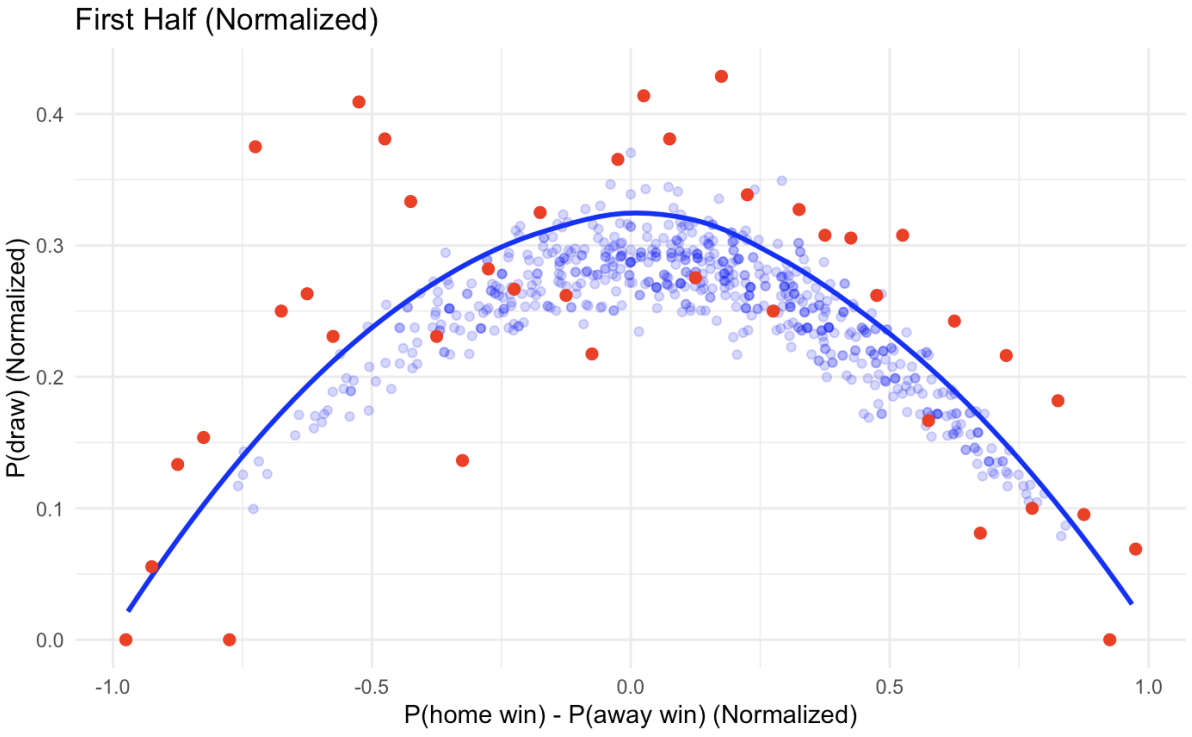
Then, I plotted both the raw data and the normalized versions for each half separately.

First Half without Normalization:

Ahmet Çeliker
2020402126

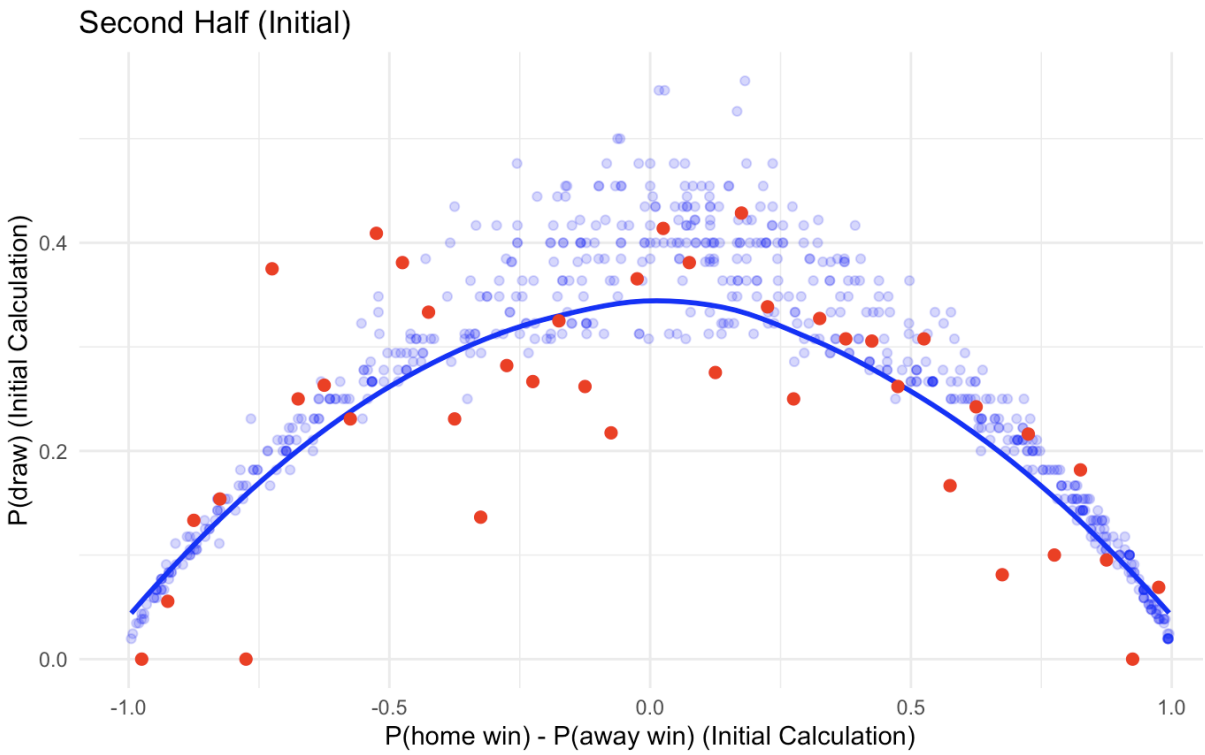


First Half with Normalization:

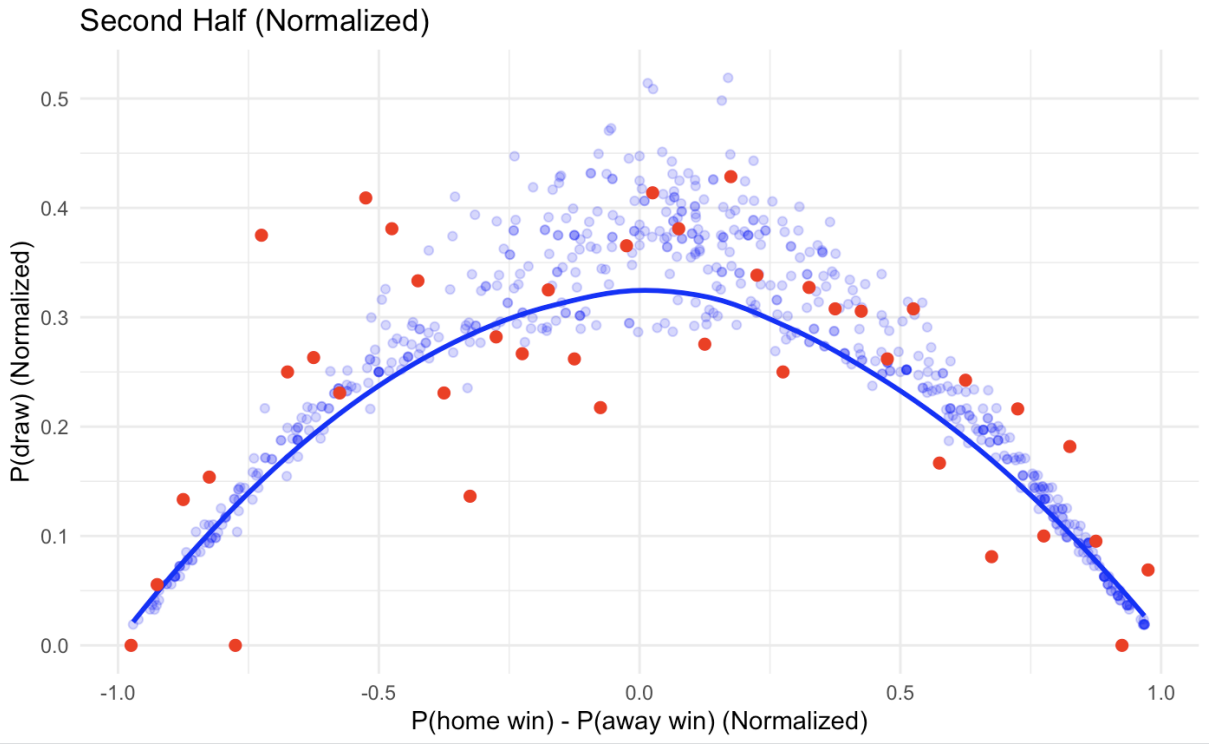


Ahmet Çeliker
2020402126

Second Half without Normalization:



Second Half with Normalization:

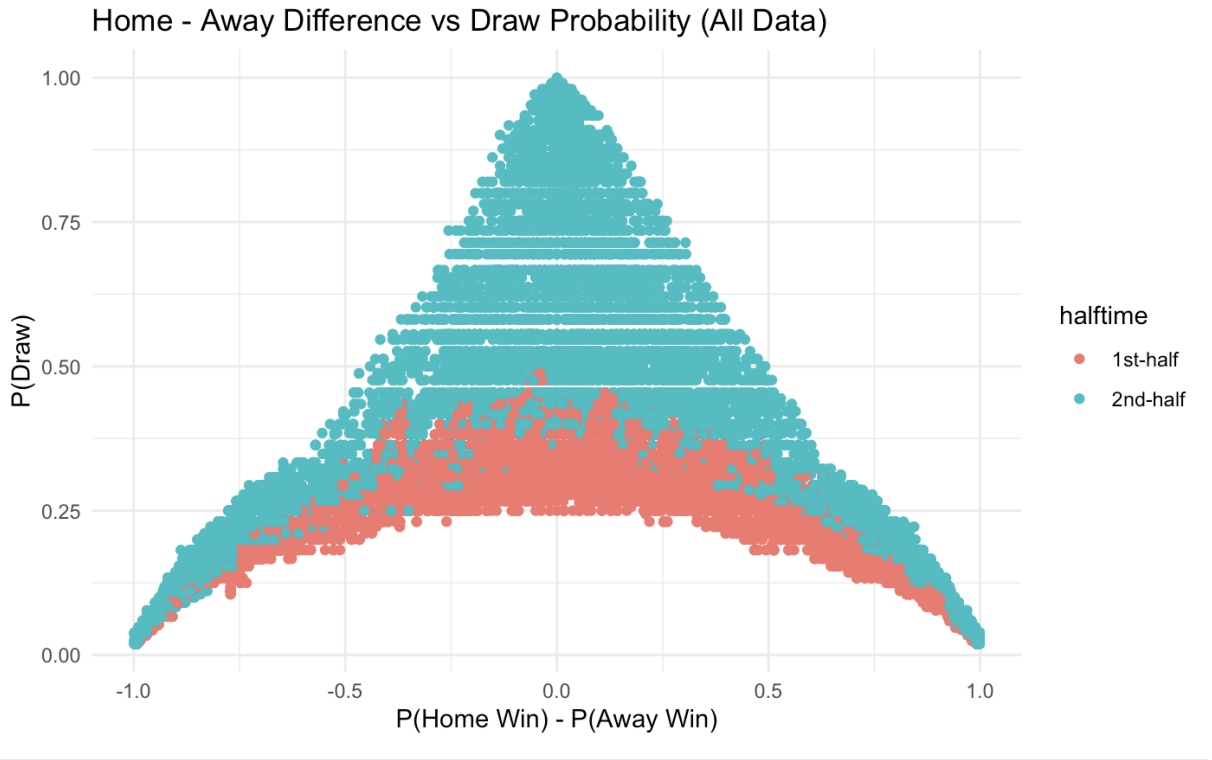


The purpose of binning the data and plotting it in this manner was to summarize and simplify the analysis by grouping the home-away probability differences into intervals. Raw data often contains a large number of individual points, which can introduce noise and make it difficult to identify clear trends. By aggregating the data into bins, I was able to reduce this noise and highlight the overall relationship between the home-away difference and the probability of a draw. Additionally, binning enabled me to compare the binned probabilities (red points) to the raw probabilities (blue points) and the smoothed LOESS curve. This comparison provided a clearer understanding of how well the aggregated probabilities aligned with the trends in the raw data.

1.2 Approach 2 : Using All Data Points

1.2.1 Odds and Probabilities Visualization

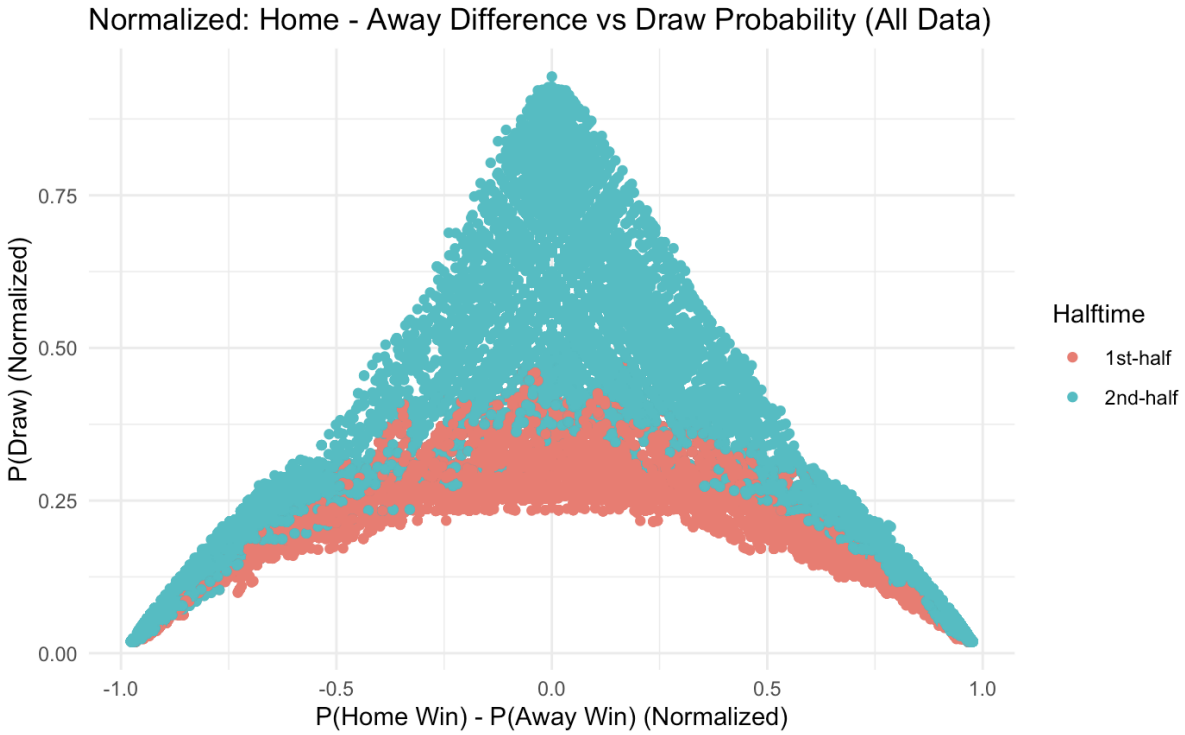
In this approach, I used **all available data points** from the dataset to analyze the relationship between the home-away probability difference and the draw probability. The visual of this plot is given below.



From the plot, it is obvious that the wider spread and higher draw probabilities in the second half indicate that bookmakers and bettors increasingly account for the match state. For

example, if a game remains tied late in the second half, the likelihood of it ending in a draw becomes more pronounced, which is reflected in the rising probabilities.

The plot using normalized probabilities is given below.



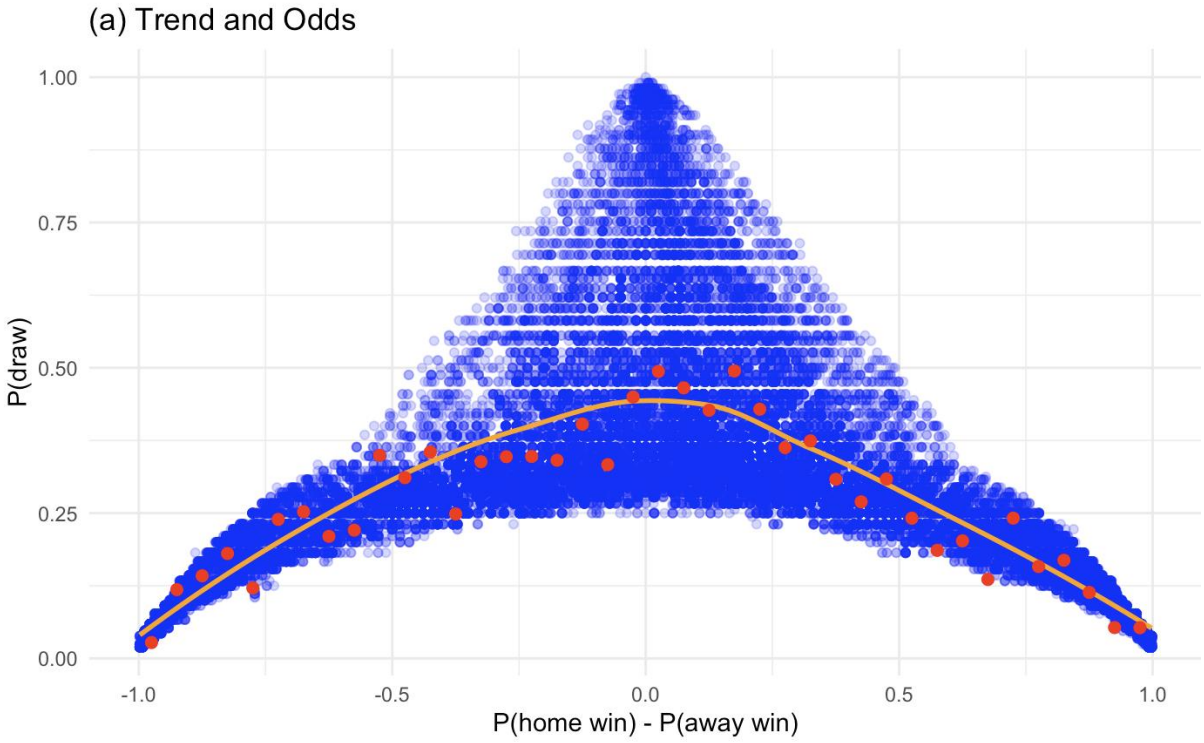
I observed a notable difference between the first and second halves. In the **first half** (red points), the draw probabilities are lower overall, and the spread of values is relatively tighter. This reflects the bookmakers' conservative approach early in the match, where odds are more stable and less influenced by real-time events. In the **second half** (blue points), the draw probabilities increase significantly near the center of the x-axis and exhibit a much wider spread. This highlights the impact of in-game events as the match progresses, particularly as the game nears its conclusion. The increasing probabilities suggest that if the score remains tied late in the match, the likelihood of a draw becomes much more pronounced.

1.2.1 Binned Data and Related Plots

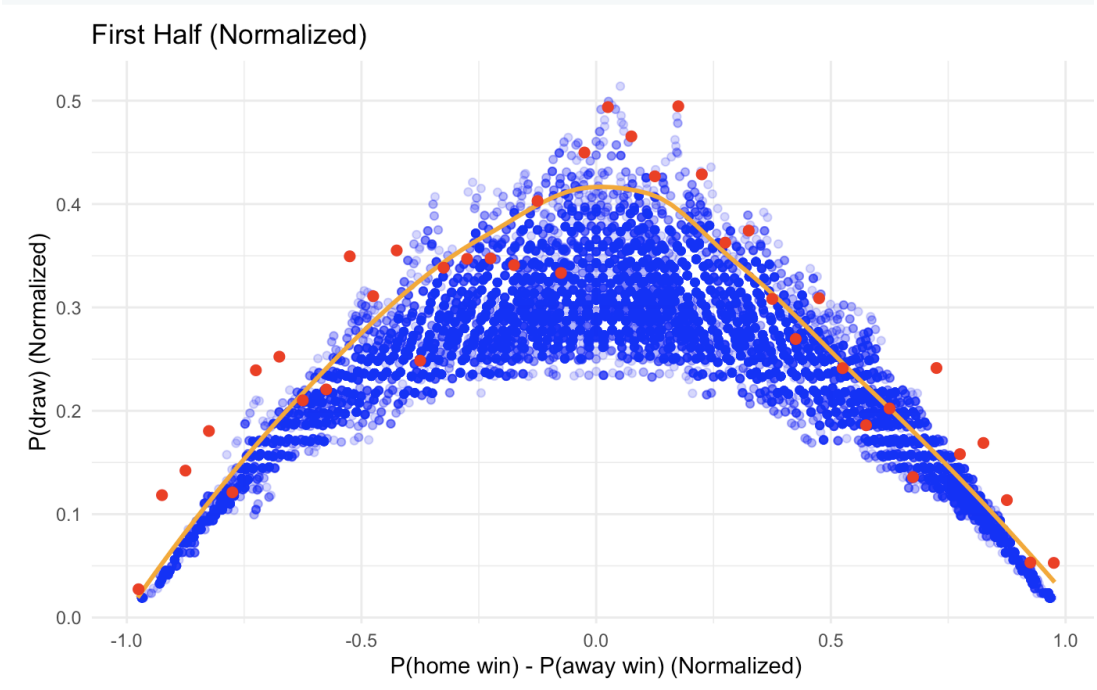
In the approach where all data points were used, I obtained the following plots after performing binning. When I binned the data at intervals of 0.05, I obtained the following graphs. In both the first-half and second-half graphs, the binned outcomes and the trend line were drawn using the entire dataset. This approach made it easier to understand the

relationship between the odds probabilities in each half and the overall trend, as well as to identify the bias.

Both Halves with Normalization:

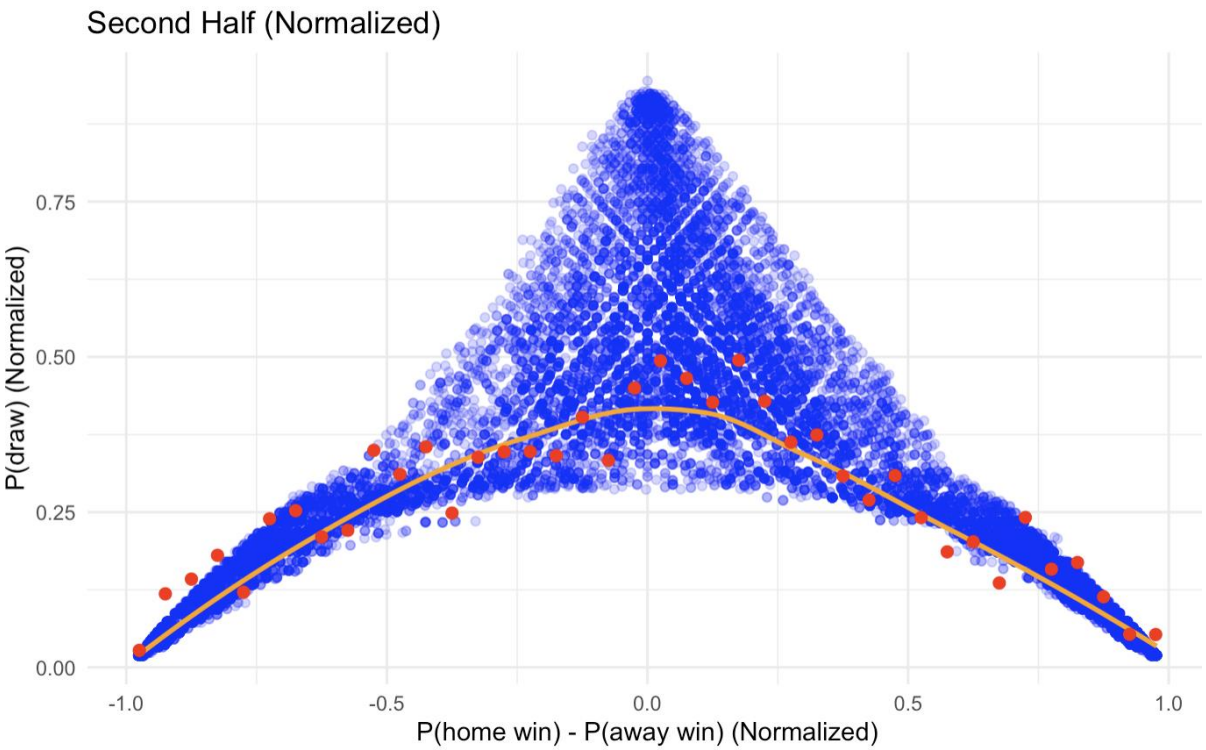


First Half with Normalization:



Ahmet Çeliker
2020402126

Second Half with Normalization:

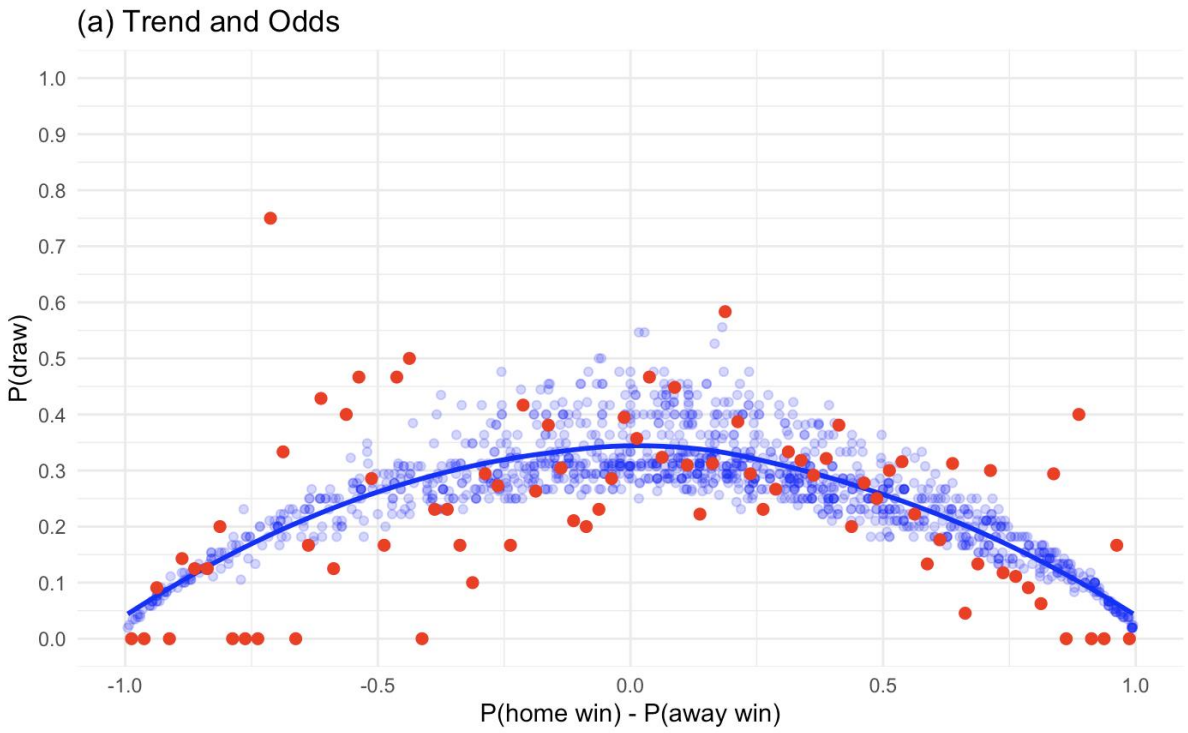


The **second half** displayed more dynamic and reactive behavior in the odds compared to the first half, introducing biases that were absent earlier in the game. This supported my earlier observation that analyzing the first timestamps of each half provided a cleaner representation of bookmakers' initial beliefs. In the **second half** (bottom plot), the bias around the zero point became much more evident. The normalized draw probabilities showed a sharp spike near (Home Win)−P(Away Win)=0, with many points exceeding 0.5 and clustering tightly. It is also clearly observed that in the graphs where the binned outcomes and the trend line are held constant (as shown above), when using first-half data, the majority of the odds probabilities lie below the trend line.

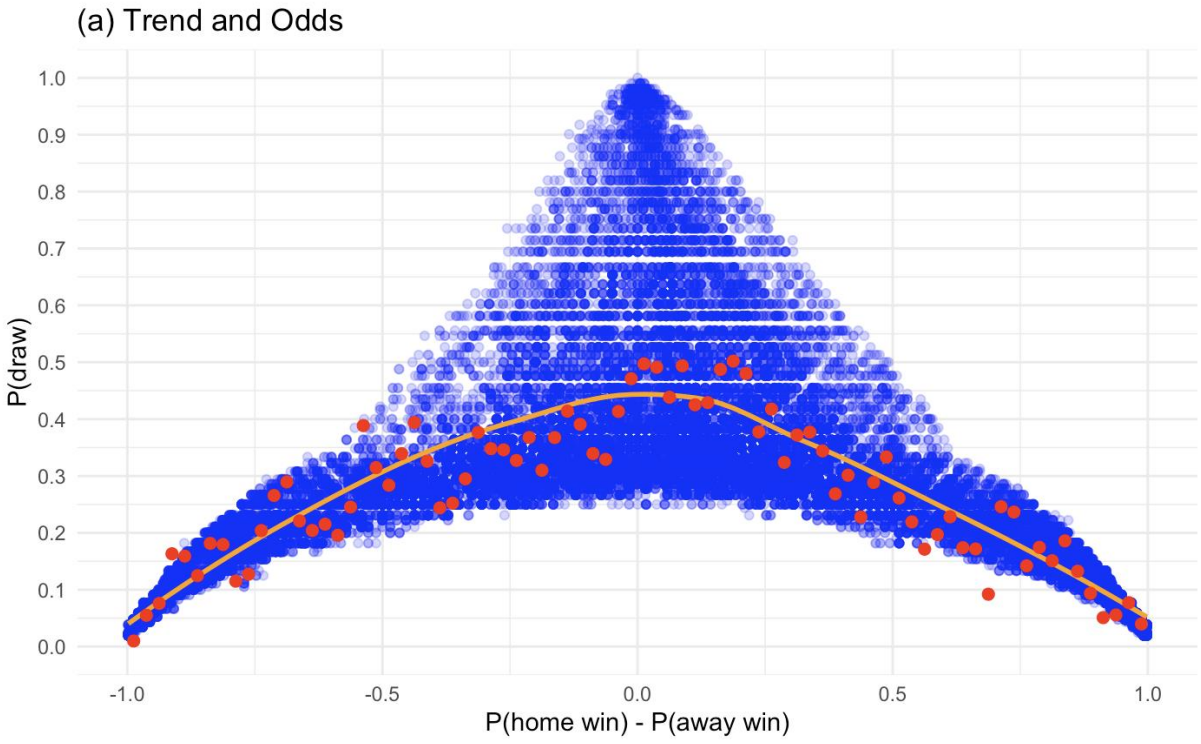
1.3 Bias Discussion

To make the bias assessment clearer, I reduced the bin intervals to 0.025 and replotted the data, obtaining the following results.

Using **Approach 1** with a bin interval of **0.025** for both halves combined, I generated the following plot.



Using **Approach 2** with a bin interval of **0.025** for both halves combined, I generated the following results.



Ahmet Çeliker
2020402126

When comparing the two plots, I observed clear differences that revealed a bias in how the odds represented the probabilities. In the first plot, where I used filtered data (e.g., the first timestamp for each half), the relationship between the home-away probability difference and the draw probability appeared more stable and well-calibrated. The LOESS trend line showed a smooth parabolic shape, and the binned averages aligned closely with the trend. Draw probabilities peaked near the center, where the match was evenly balanced, and decreased symmetrically as the home-away difference moved toward the extremes. The probabilities remained conservative, not exceeding 0.4, suggesting that bookmakers' odds in the early stages of the match reflected rational expectations with minimal bias. Specifically, near the point where $P_{home} - P_{away} \approx 0$, the binned outcomes stayed above the trend line, indicating an increasing bias.

In the second plot, where I used all available data points, significant deviations and bias emerged. The draw probabilities spiked dramatically, reaching as high as 1.0 when the home-away difference was close to zero. This was likely caused by real-time adjustments as the match progressed, particularly in later stages when the score remained tied. Bookmakers dynamically updated their odds to reflect the increasing likelihood of a draw, inflating the probabilities. The LOESS curve became less smooth due to the higher variability, and while the binned averages still followed the general trend, they showed greater deviations caused by the noisy data. Overall, the binned outcomes appear to lie below the trend line. While 46 of the binned outcomes are below the trend line, 28 are above it, and the remaining ones are almost exactly on the trend line. When approaching from the left side of the $P_{home}-P_{away}$ axis (in the range $-0.5<P_{home}-P_{away}<0$), approximately 16% of the binned outcomes are above the trend line, while the rest lie below it, which reflects high bias around center.

Overall, the second plot demonstrated a clear bias introduced by in-game events and real-time odds adjustments, particularly near the end of matches. In contrast, the first plot, based on filtered data, provided a cleaner view of bookmakers' initial beliefs, minimizing the influence of evolving match dynamics. This comparison highlighted the importance of focusing on the early-match or halftime odds to analyze systematic biases without interference from real-time outcomes.

Ahmet Çeliker
2020402126

2. Task 2

In this task, I applied several filtering and transformation steps to the dataset to analyze specific scenarios involving match states, state changes, and red cards. When all matches were analyzed, **101 out of 648 matches** had a red card shown.

2.1 First Filtering Cases

I systematically filtered and processed the data to focus on specific scenarios:

1. I identified matches with or without a recorded 90th minute.
2. I examined matches where the state of play changed at or after the 90th minute or final available minute.
3. I analyzed the impact of early red cards on the match data and excluded such matches for cleaner analysis.

First, I flagged matches that had a recorded **90th minute** and merged this information back into the dataset. This allowed me to differentiate between matches with and without the 90th minute. For matches **without the 90th minute**, I identified the latest available state and compared it to the state recorded at the 90th minute for the other matches.

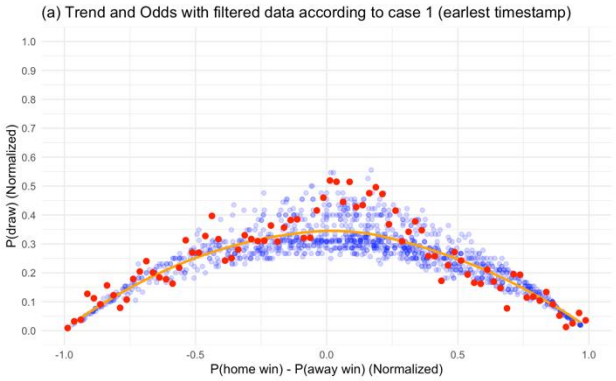
Next, I focused on matches where a **state change** occurred. For matches with a 90th minute, I checked whether the state at the 90th minute differed from the final result. For matches without a 90th minute, I compared the latest recorded state to the result. This helped isolate instances where significant changes occurred in the outcome.

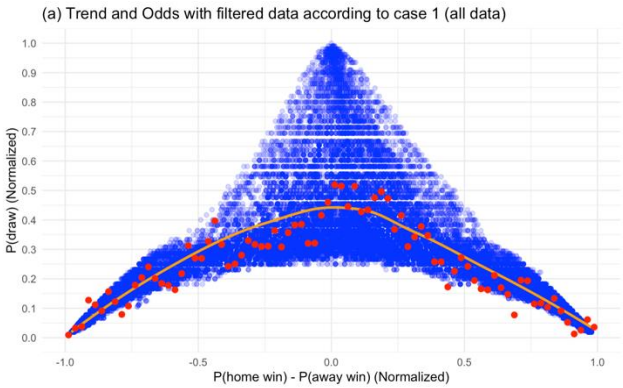
Additionally, I analyzed the impact of **early red cards** by identifying matches where a red card was issued within the first 15 minutes.

As a result of all these filtering steps, I identified **41 matches** where a **state change** occurred at or after the 90th minute. Additionally, there were only **2 matches** where a **red card** was issued within the first 15 minutes of the game.

After the filtering, where noise was removed, I obtained two plots: one using the **earliest timestamp row for each half** and another using **all data after filtering**. The resulting plots are shown below.

(I used normalized data and binning interval is 0.025)





After applying filters to exclude matches with early red cards (first 15 minutes) and matches with state changes at or after the 90th minute, the data appears cleaner and more concentrated. The LOESS curve (yellow line) is smoother, with fewer extreme outliers and irregularities. This suggests that the remaining data better represents stable and predictable match dynamics. Before filtering, the data includes **noise** introduced by extreme scenarios, such as early red cards and late-game state changes.

The **binned data points** (red points) around the **center of the x-axis** (P(Home Win) - P(Away Win) ≈ 0) show noticeable deviations from the trend line. This suggests a **potential bias** in the filtered data. The red binned data points around the center (P(Home Win) - P(Away Win) ≈ 0) appear **above the trend line**, indicating that the actual draw probabilities in this range are **higher than expected** by the model.

One of the comment I can make is that removing volatile matches (e.g., state changes at 90 minutes) reduces noise but may also amplify bias in cases where match dynamics are naturally balanced.

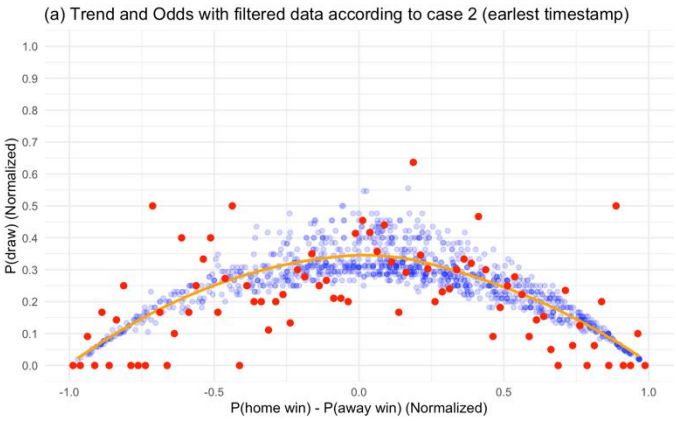
2.2 Second Filtering Cases

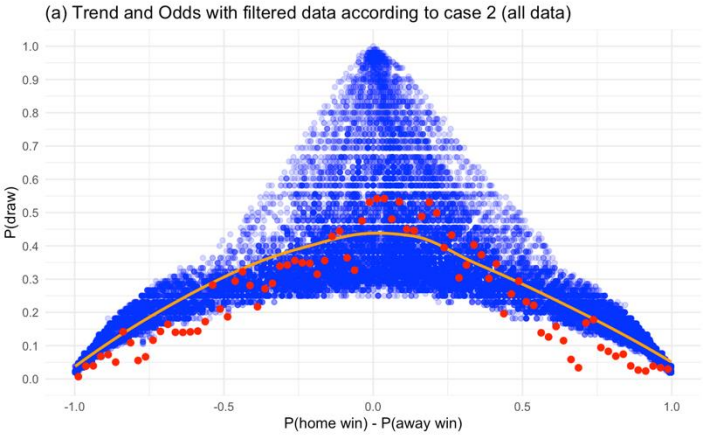
In **Case 2**, I applied a slightly modified filtering process compared to **Case 1** to refine the dataset further. The key difference lay in focusing on the **80th minute** instead of the 90th

minute to detect state changes and removing matches with early red cards as before. I identified matches where a red card was issued within the **first 30 minutes** of play.

After applying all the filtering steps, I found **92 matches** with a state change occurring at or after the **80th minute**. Furthermore, there were **18 matches** where a red card was shown within the **first 30 minutes** of play.

(I used normalized data and binning interval is 0.025)





In Case 2, the bias significantly increased across the entire dataset. When I selected the earliest timestamps, the binned outcomes generally exhibited a high bias around the trend line throughout the entire x-axis. However, when using all available data, the binned outcomes predominantly fell below the trend line, except for values around the center of the x-axis (P(Home Win) - P(Away Win) ≈ 0). At this central point, I observed binned outcomes above the trend line, indicating a deviation in this region compared to the overall trend.

2.3 Third Filtering Cases

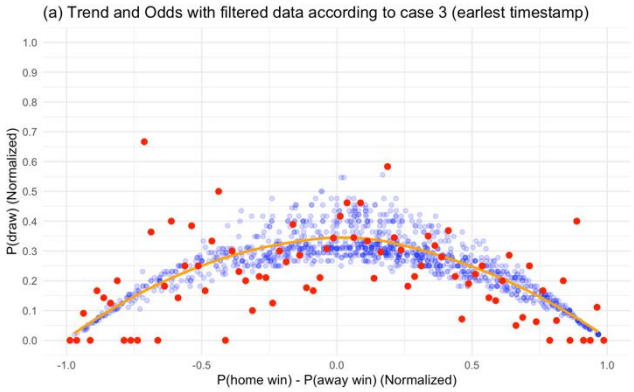
This filtering step was applied **on top of Case 1**. In this additional filtering step, I specifically targeted matches where **penalties occurred at or after the 80th minute** and influenced the result of the game. I achieved this by calculating the difference in the penalty count for home and away teams between consecutive timestamps. If a penalty was observed (i.e., the count increased) at or beyond the 80th minute, the corresponding match was flagged and excluded from the final dataset. In Case 1, I had already excluded matches where a state

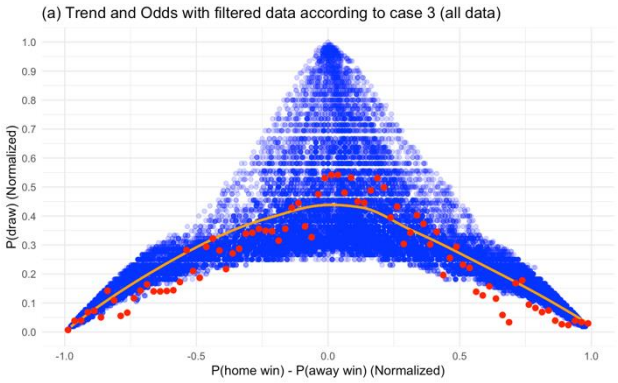
change occurred at or after the **90th minute** and games with **early red cards** within the first 15 minutes. The additional penalty filter further refined the dataset by removing matches where penalties awarded late in the game could have directly influenced the result.

In this case, I identified **32 matches** where a penalty occurred at or after the **80th minute** and directly influenced the outcome of the game. A total of **71 matches** were removed from the dataset. There are **4 matches** that overlap, where both a penalty occurred **after the 80th minute** and a goal was scored **after the 90th minute**.

The plots are given below.

(I used normalized data and binning interval is 0.025)





Case 2 and Case 3 displayed very similar plots. In both cases, when I used the earliest timestamp data, the bias was quite high and consistently observed across the x-axis. When the full dataset was used, I observed binned outcomes above the trend line near the region where $P(\text{Home Win}) - P(\text{Away Win}) \approx 0$, whereas the outcomes in other areas tended to fall below the trend line. However, it was clear that Case 2 and Case 3 exhibited very similar characteristics in terms of bias.

2.4 General Results

In this analysis, I applied multiple filtering steps across three distinct cases to investigate match dynamics, reduce noise, and analyze the relationship beetween $P(\text{Home Win}) - P(\text{Away Win})$ and P(Draw). Each case progressively refined the dataset by

addressing factors that could disproportionately influence match outcomes, such as early red cards, late state changes, and penalties.

Removing extreme scenarios (e.g., early red cards, late penalties, and state changes) reduced noise in the dataset but amplified bias in specific regions. Matches with $P(\text{Home Win}) - P(\text{Away Win}) \approx 0$ consistently demonstrated higher observed draw probabilities compared to the model's expectations, highlighting a potential inefficiency in the implied probabilities.

Although reducing noise resulted in a smoother trend line and a more stable distribution of probabilities around the trend line, it also led to an increase in bias, particularly around the center of the x-axis ($P(\text{Home Win}) - P(\text{Away Win}) = 0$). The increase in bias can be attributed to the removal of noise, which revealed that the odds set by bet makers are lower than the observed binned outcomes.

3. Task 3

In this task, I built a decision tree model to predict the outcome of a game using match statistics available for each minute as features. The primary objective was to identify patterns in the match data that could help foresee the match result and assess the efficiency of the odds market by comparing predicted probabilities with the bookmakers' implied probabilities.

3.1 Filtering and Cleaning

I filtered out suspended or stopped matches to ensure the data only contained valid instances. Missing values in numeric columns were handled using a combination of forward filling, backward filling, and zero-imputation where necessary.

I applied the following conditions to fill in missing numeric values:

- **Condition 1**: If **all values** in a column were missing (NA), I replaced the entire column with zeroes. This ensured that completely missing columns did not disrupt the analysis.
- **Condition 2**: If there were missing values at the **start** of a sequence, I filled them with zeroes. This step addressed incomplete data at the beginning of a match where statistics may not have been recorded yet.

- **Condition 3**: I used **forward filling** to propagate the last observed value to subsequent missing entries. This accounted for situations where values were missing intermittently but were expected to persist over time.
- **Condition 4**: I applied **backward filling** to propagate future non-missing values backward. This step filled gaps that were preceded by missing data but followed by observed values.
- **Condition 5**: For any remaining missing values after forward and backward filling, I filled them with the **last non-NA value** in the column. This final step ensured that all entries had valid values even if gaps remained after other imputation methods.

In this way, I evaluated each match on its own and also set the events from the start of the match to the status change in the feature to 0.

3.2 Feature Derivation and Feature Selection

I created a **"total_minute"** feature to represent the cumulative match time across the first and second halves.

As part of the feature selection process, I removed several columns that were deemed **redundant** or **inherently related** to other features. Many of these columns (e.g., minute, second, current_time) contained information that was already captured or derivable from other features such as **"total_minute"**. Features like name, fixture_id, and timestamps (e.g., half_start_datetime) were identifiers or metadata, which do not contribute directly to predicting the match result. These were unnecessary for the decision tree and would add noise. Columns such as Score.Change...away and Score.Change...home were implicitly represented by other features like odds, match states, and cumulative statistics at each minute. Their removal ensured that the model only relied on relevant and non-duplicative information.

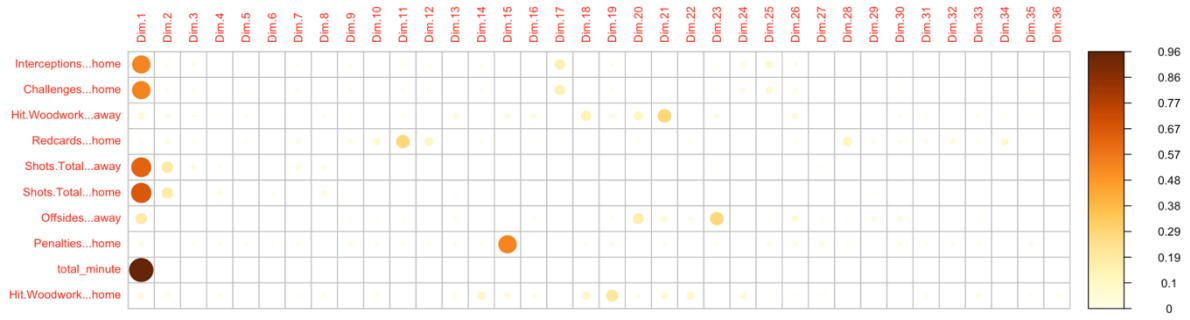
The **"result"** column, representing the outcome of the match, was converted into a factor variable to enable classification. Draws were encoded as 0, while home and away wins retained their respective categorical values.

I converted the **halftime** column, which originally contained categorical values ("1st-half" and "2nd-half"), into numerical representations. I assigned 1 to represent the first half and 2 for the second half, while leaving any unrecognized or missing values as NA. This step allowed the model to treat **halftime** as an ordinal feature, ensuring it could assess its impact on the match outcome. I also transformed the **current_state** column into a numeric format.

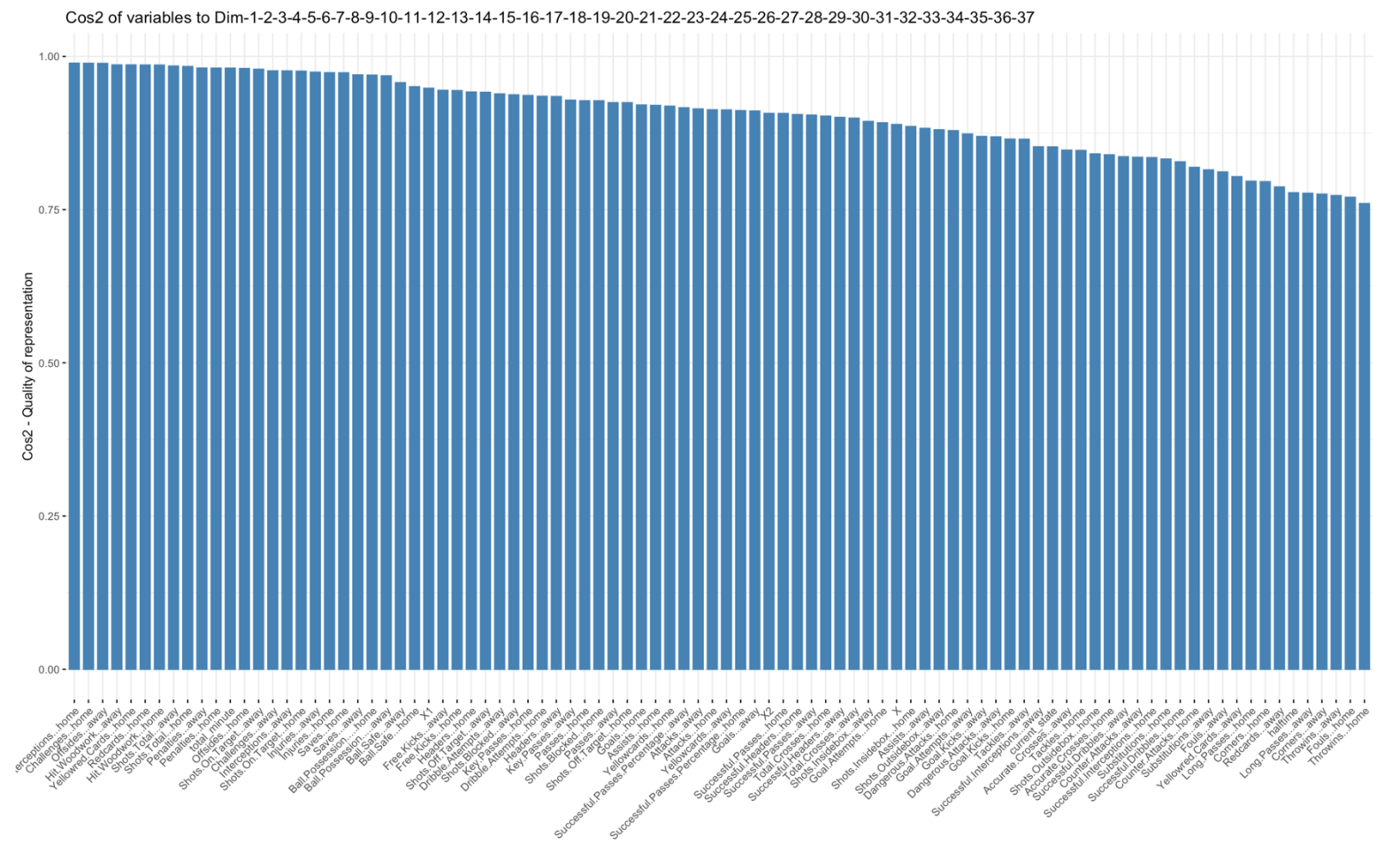
Ahmet Çeliker
2020402126

3.3 Decision Tree with Principal Component Analysis (PCA)

I performed **PCA** to reduce dimensionality and identify the most significant features contributing to variance in the dataset. I selected enough principal components to explain at least 90% of the variance. The number of principal components (PCs) covering 90% of the total variance is 37, reducing the new dimensionality to 37. The top 10 features contributing the most to these PCs are shown in the graph below.



Another representation is provided below. As can be seen, no single feature dominates significantly. The impact of the features decreases gradually.

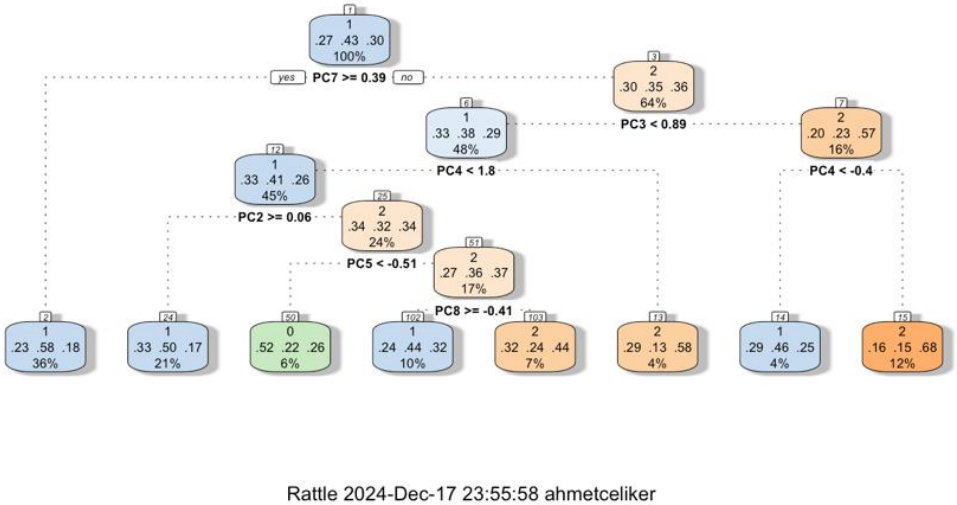


As observed from the visualizations above, certain features stood out as having a significant impact on the PCA model. For **PC1**, which explained the highest variance with %40.5 of the total variance, the most influential features were **total_minute**, **shots.total...home**, **shots.total...away**, **challenges...home**, and **interceptions...home**. For the **37 principal components** used in the model, predictors such as **offside...away**, **hit.woodwork...away**, **redcards...home**, and **yellowcards...home** also proved to be important.

3.3.1 Base Model for Decision Tree Using Principal Components

I created a decision tree model using the principal components as features. My main motivation was to reduce the high dimensionality of the data and break features with high inter-correlation (to prevent multicollinearity) from the original 106 features. Initially, I built a baseline model using the rpart function in R without performing any parameter tuning. The default parameters of the rpart function were as follows: minsplit = 20, minbucket = 7, cp = 0.01, maxdepth = 30, xval = 10.

When I trained the model on the entire dataset without performing any train-test split, the result was as follows



Rattle 2024-Dec-17 23:55:58 ahmetceliker

Ahmet Çeliker
2020402126

The decision tree shown in the image has a high level of interpretability due to its relatively shallow depth and clear branching structure. Each split is based on principal components (PCs), making it straightforward to follow and understand how the model makes decisions at each node. This simplicity allows me to clearly see the conditions under which predictions are made and which features (PCs) have the most influence.

However, despite this high interpretability, the accuracy of the model is not at the desired level.

When I split the dataset into 70% for training and 30% for testing to evaluate thedecision tree model, I obtained the following confusion matrix results. This evaluation provides a clearer understanding of the model's generalization performance on unseen data.

The result of confusion matrix is given below.

```
Overall Statistics

               Accuracy : 0.5464
                 95% CI : (0.5388, 0.5539)
    No Information Rate : 0.43
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2551

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

The model correctly predicts 54.64% of the outcomes. While better than random guessing (No Information Rate of 43%), the accuracy remains moderate, indicating room for improvement. Kappa measures agreement between predicted and actual classes, adjusted for chance. A Kappa of 0.2551 suggests fair agreement but far from strong performance. The accuracy is statistically significant, with a very low p-value (<2.2e-16). This means the model's performance is better than random guessing. The model only correctly identifies **12.42%** of actual Class 0 instances. This is very low, meaning the model misses a large proportion of true Class 0 matches. It identifies **87.93%** of Class 1 matches correctly. This is the strongest performance across all classes. Many predictions fall into Class 1, which could be a reflection of the underlying imbalance in the data (Class 1 has the highest prevalence: 43%).

This is my base model, so by tuning these parameters and addressing class imbalance, the decision tree can be optimized for better sensitivity and overall accuracy across all classes.

To evaluate and tune my decision tree model using PCA-transformed features, I applied cross-validation through the train function in R. Specifically, I trained the model with method = "rpart", which implements a CART (Classification and Regression Tree) approach. Using cross-validation allowed me to obtain a more reliable and generalizable performance estimate for the decision tree. The process involved **bootstrapped resampling with 25 repetitions** on a dataset containing **39,288 samples, 37 predictors**, and a target variable with **3 classes** (‘0’, ‘1’, ‘2’).

The results of the Cross Validation model with default parameters of function train in Caret package is given below.

```
CART

39288 samples
   37 predictor
    3 classes: '0', '1', '2'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 39288, 39288, 39288, 39288, 39288, 39288, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.01711349  0.5157230  0.20994886
  0.02980340  0.4904118  0.15333874
  0.04577748  0.4649917  0.08650865
```

Based on these results, I selected **cp = 0.0171** as the optimal value for the model, as it provided the best balance between complexity and accuracy. This parameter allowed the decision tree to capture meaningful patterns in the data without overfitting.

3.3.2 Upgraded Model with Class Weights

I trained a weighted decision tree model, where I incorporated a loss matrix to address the class imbalance issue in the dataset. The loss matrix allowed me to penalize the misclassification of specific classes more heavily, particularly Class 0 and Class 2, which were underrepresented compared to Class 1.

I used matrix(c(0, 5, 5, 2, 0, 2, 3, 3, 0), nrow = 3 for LOSS matrix. This structure prioritizes the correct classification of Class 0 by assigning a higher penalty when Class 0 instances are misclassified. The penalties for misclassifying Class 2 are also moderately high, while Class 1, being the dominant class, has lower misclassification costs.

The result is given below.

```
Overall Statistics

               Accuracy : 0.5137
                 95% CI : (0.5061, 0.5213)
    No Information Rate : 0.43
    P-Value [Acc > NIR] : < 2.2e-16
```

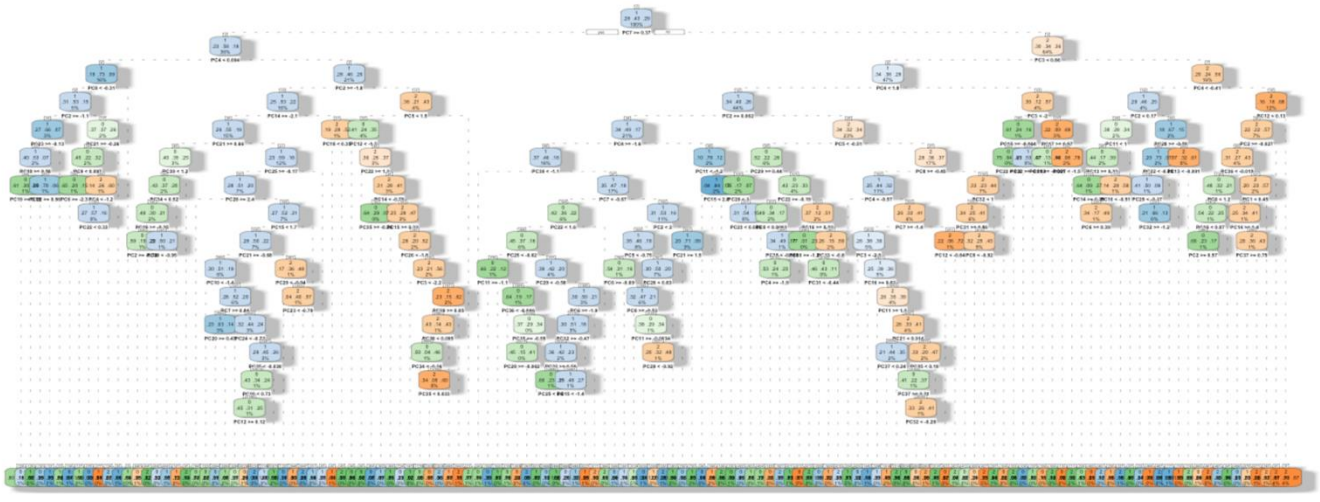
```
                  Kappa : 0.2186

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

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.0000   0.7631   0.6177
Specificity            1.0000   0.5295   0.6882
Pos Pred Value            NaN   0.5502   0.4597
Neg Pred Value         0.7304   0.7477   0.8074
Prevalence             0.2696   0.4300   0.3004
Detection Rate         0.0000   0.3281   0.1856
Detection Prevalence   0.0000   0.5963   0.4037
Balanced Accuracy      0.5000   0.6463   0.6530
```

When comparing the results of the weighted decision tree model to the base model, it became evident that the overall performance did not improve significantly. The **accuracy** of the weighted model remained at **51.37%**, which is similar to the base model. The **Kappa statistic** was **0.2186**, indicating that the model still struggles to predict beyond chance, especially for the minority classes.

3.3.3 Grid Search for Parameters

In this step, I conducted an extended grid search to tune multiple hyperparameters of the rpart decision tree model, specifically cp, minsplit, and maxdepth. I defined a parameter grid that included combinations of the complexity parameter (cp) ranging from **0.001 to 0.02**, the minimum number of observations required to attempt a split (minsplit) at **10, 20, and 30**, and the maximum allowable depth of the tree (maxdepth) at **5, 10, and 15**.

The decision tree looks like that:

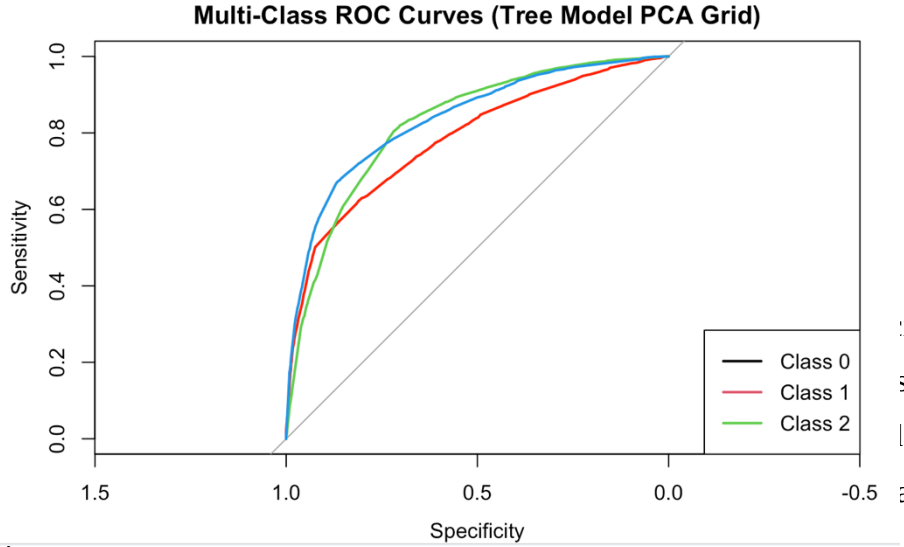


```
Overall Statistics

               Accuracy : 0.6861
                 95% CI : (0.6791, 0.6932)
    No Information Rate : 0.43
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5087

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

The **best model** achieved an accuracy of **0.686**, which represents a significant improvement compared to the earlier models.

To determine whether the model was overfitting, I examined the training and testing errors. If there is a significant difference between these two errors, it indicates that the model has overfitted.

The test error was 0.3139, while the training error was 0.2940, indicating that these two values are relatively close to each other. This suggests that the model is not significantly overfitting, as the performance on the training data is comparable to its performance on the test data.



The mul ision tree model optin ss is calculated s l against all others. The ative ability for each class.

- **Class 0**: 0.7835
- **Class 1**: 0.8241
- **Class 2**: 0.8361

The results show that Class 2 (green curve) achieved the highest AUC, indicating the strongest predictive performance for distinguishing this class. Class 1 (red curve) follows closely with an AUC of 0.8241, reflecting good balance between sensitivity and specificity. However, Class 0 (blue curve) achieved the lowest AUC at 0.7835, suggesting that the model struggles more to differentiate this class, likely due to class imbalance or overlapping features.

3.4 Feature Selection Using Lasso Regression

In this step, I implemented feature selection using LASSO regularization to reduce the dimensionality of the dataset and eliminate irrelevant predictors. First, I removed unnecessary columns, such as identifiers and timestamps, that did not contribute meaningful information to the model.

To perform feature selection, I converted the predictors into a matrix (x) and the target variable into a numeric vector (y). I applied LASSO regression using the cv.glmnet() function with alpha = 1, which imposes an L1 penalty to shrink coefficients and enforce sparsity.

The optimal lambda (lambda.min) minimized the mean cross-validated error, while the stricter lambda.1se provided a more conservative subset of features. When the optimal lambda value was 9.374985e-05, the first features that were shrunk to zero (i.e., with s1 value of 0) were: **Challenges...away, Interceptions...home, Total.Crosses...away.**

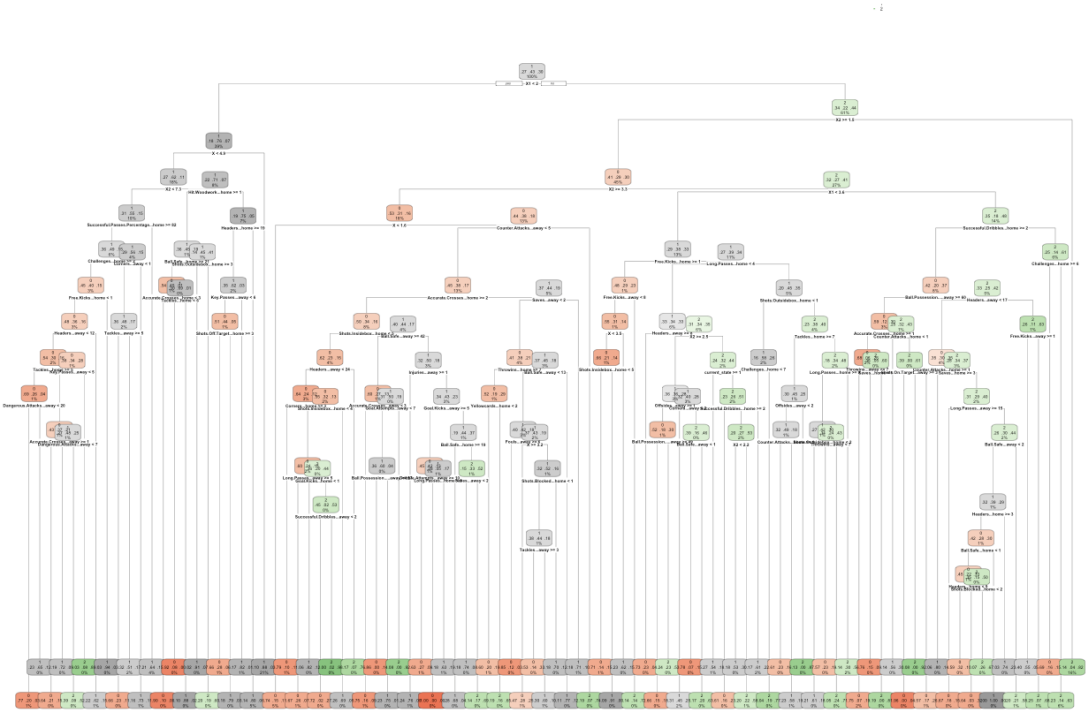
When I applied the **LASSO** regularization with the stricter penalty parameter **lambda.1se**, the following features were shrunk to zero and eliminated from the model: **Attacks...home**, **Challenges...away**, **Dribble.Attempts...home**, **Interceptions...away**, **Interceptions...home**, **Passes...home**, **Shots.Insidebox...away**, **Shots.Total...home**, **Successful.Headers...away**, **Successful.Passes.Percentage...away**, **Total.Crosses...away**

These features were deemed the least significant under the stricter regularization penalty, as their coefficients were reduced to zero, indicating minimal contribution to the model's predictive performance.

Ahmet Çeliker
2020402126

From the LASSO results, I extracted the non-zero coefficients corresponding to the most significant predictors. I filtered out the intercept term and retained the selected features, ensuring that only the most impactful variables were included in the next modeling step.

However, due to LASSO's **linear** nature, some **non-linear relationships** and feature interactions were ignored. Moreover, certain features that might have provided critical splits in the decision tree but had low global importance were eliminated.
The decision tree built using the data obtained after feature selection with LASSO is shown below.



I trained a decision tree model using the reduced feature set with the rpart() function, setting the control parameters to cp = 0.01, minsplit = 10, and maxdepth = 15. These parameters were previously found to be optimal for the PCA-based model. My motivation for keeping the parameters fixed was to identify which model—PCA or LASSO-based feature selection—performs better under identical conditions.

The resulting confusion matrix is given below.

```
Overall Statistics

               Accuracy : 0.6861
                 95% CI : (0.6791, 0.6932)
    No Information Rate : 0.43
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5087

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

```
Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.5011   0.8120   0.6721
Specificity            0.9242   0.7109   0.8660
Pos Pred Value         0.7094   0.6793   0.6830
Neg Pred Value         0.8338   0.8337   0.8601
Prevalence             0.2696   0.4300   0.3004
Detection Rate         0.1351   0.3491   0.2019
Detection Prevalence   0.1905   0.5139   0.2956
Balanced Accuracy      0.7127   0.7615   0.7691
```

I then applied cross-validation (CV) to the model to ensure its performance was robust and to evaluate its generalization capability on unseen data.

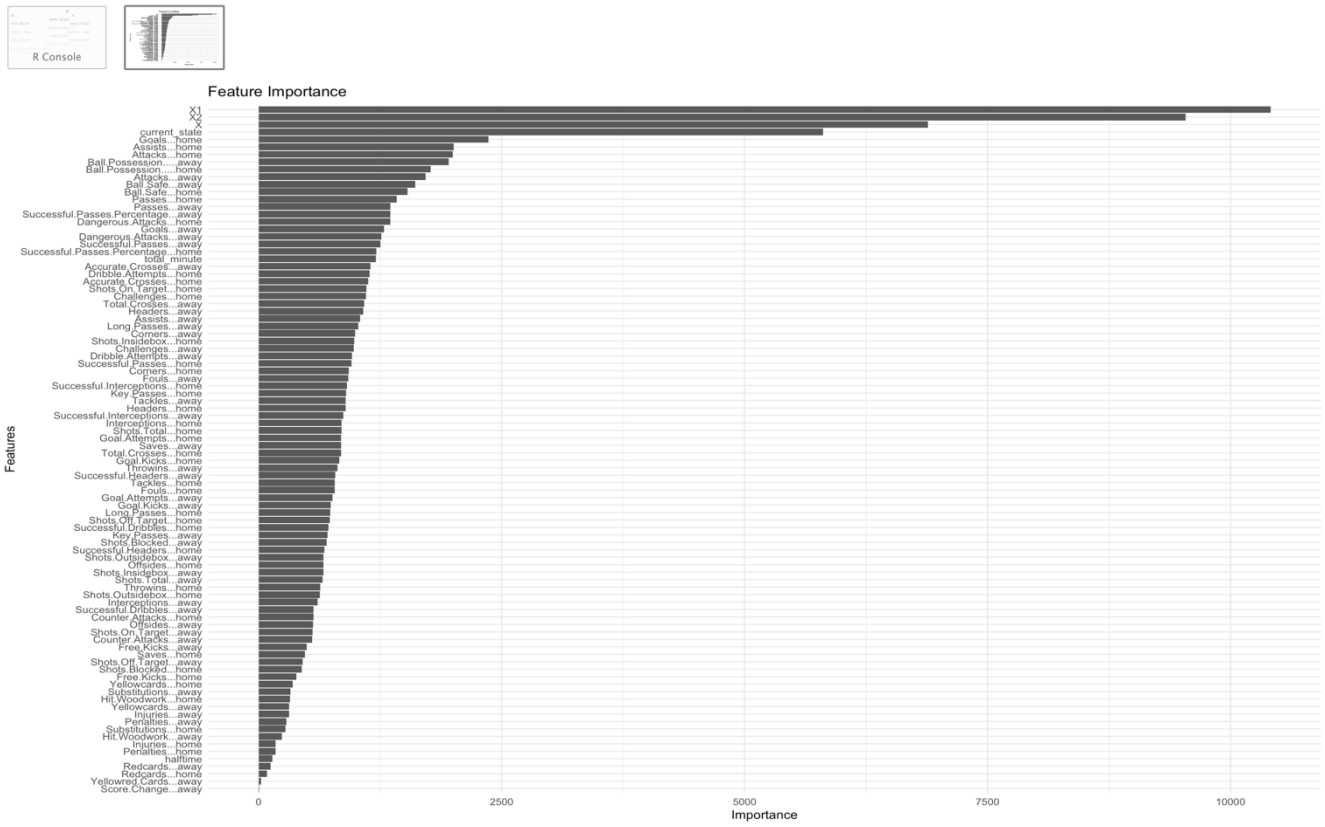
```
CART

39288 samples
   37 predictor
    3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 35359, 35359, 35359, 35360, 35359, 35360, ...
Resampling results across tuning parameters:

  cp     Accuracy   Kappa
  0.001  0.6902107  0.5159858
  0.003  0.5943030  0.3604499
  0.005  0.5660763  0.3080941
  0.007  0.5520006  0.2798796
```

As seen above, when the same parameters were used, the results show almost no difference between the model using features obtained through PCA and the one using features obtained through LASSO. This method did not achieve any significant improvement.

3.5 Decision Tree with Full Data

In this case, I trained a decision tree model using the **full dataset** that included all features, without applying any feature selection methods like **LASSO** or **PCA**. The full model achieved an accuracy of **0.74**, which outperformed the previous models that used reduced feature sets. This improvement indicates that the additional features in the full dataset contributed valuable information, enabling the model to better capture patterns and relationships within the data. Decision trees are particularly effective at capturing such **non-linear relationships**, which LASSO and PCA failed to account for. This is why the full model achieved higher accuracy, while the simplified feature sets led to a loss of predictive performance.

The model trained on the entire dataset (after removing only irrelevant and inherently correlated features) without a train-test split shows the feature importance as follows.

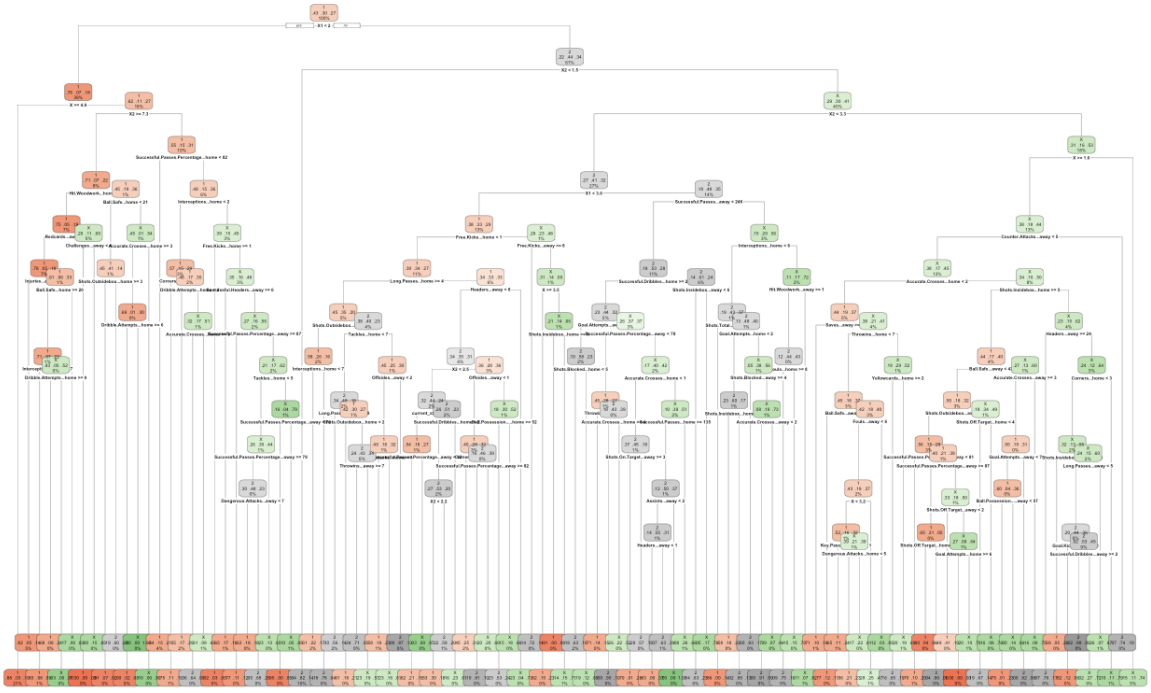


The features with the highest importance in the model are X1, X2, X (betmaker odds), current_state, Goals...home, Assists...home, Attacks...home, Attacks...away, Ball.Possession.....away, Ball.Possession.....home, Ball.Safe...away, and Ball.Safe...home.

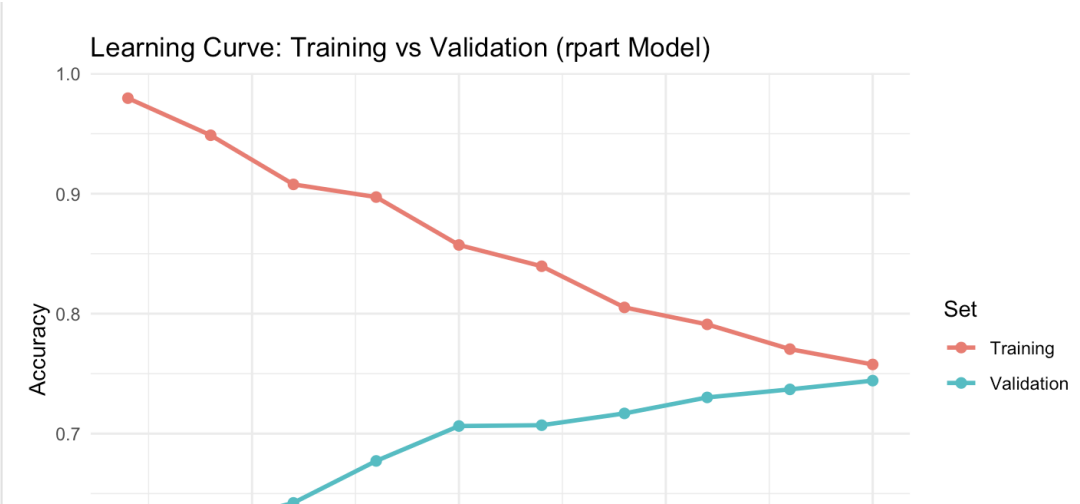
The high importance of **X1, X2**, and **X** (betmaker odds) in the model is justified because they are derived from a combination of comprehensive historical and real-time factors. Also, Odds indirectly capture multiple aspects of the match, including **team form, player availability, historical matchups, weather conditions**, and other unseen variables. Therefore, odds serve as **aggregated features** that encapsulate information from numerous other predictors.

In this scenario, I kept the model parameters the same across all cases to ensure a **fair comparison** of the models. Specifically, I used the same cp, minsplit, and maxdepth values (cp = 0.01, minsplit = 10, and maxdepth = 15) for consistency. The decision tree trained on the **full feature set** produced the following results and structure:

Ahmet Çeliker
2020402126



```
Overall Statistics

               Accuracy : 0.7397
                 95% CI : (0.733, 0.7464)
    No Information Rate : 0.4302
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5969

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

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.5506   0.8404   0.7687
Specificity            0.9065   0.8072   0.8829
Pos Pred Value         0.6898   0.7669   0.7336
Neg Pred Value         0.8422   0.8701   0.9010
Prevalence             0.2742   0.4302   0.2955
Detection Rate         0.1510   0.3616   0.2272
Detection Prevalence   0.2189   0.4715   0.3097
Balanced Accuracy      0.7285   0.8238   0.8258
```



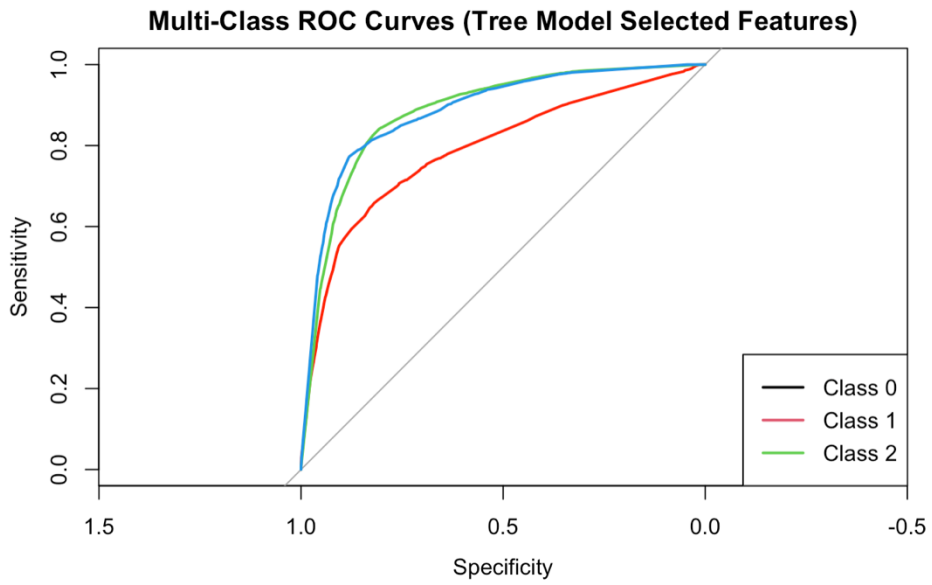
This model appears to be the best option in terms of achieving both high accuracy and better interpretability.

As can be seen from the graph above, as the training set size increases, the training annd validation accuracy rates converge. I interpreted this as the model not overfitting.

3.5.1 Feature Selection Based on Decision Tree Results

I focused on refining the decision tree model by selecting only the most important features. My goal was to simplify the model, enhance interpretability, and ensure efficient performance without sacrificing predictive accuracy. To accomplish this, I identified the most significant features from a previously trained decision tree model by filtering out predictors with zero importance. The features **Halftime**, **Assists...away**, **Score.Change...away**, **Score.Change...home**, and **Yellowred.Cards...home** had an importance score of **0** in the decision tree model.

When I compared this ROC curve and the corresponding AUC values (0.789 for Class 0, 0.882 for Class 1, and 0.884 for Class 2) to the PCA and LASSO-based decision tree models, I observed that the model trained on the full dataset achieved the best overall performance.



Unlike PCA and LASSO, which may have removed predictors that were not globally significant but still influential in certain splits, this approach ensured no critical information was lost.

3.6 Parameter Tuning on the Last Model

3.6.1 Grid Search

To further optimize the decision tree model, I performed a grid search over key hyperparameters, including the complexity parameter (cp), the minimum number of observations required to split a node (minsplit), and the maximum depth of the tree (maxdepth). The grid search explored a range of values:

- cp: 0.001 to 0.02 (in increments of 0.002),
- minsplit: 5 to 50 (in increments of 10),
- maxdepth: 5 to 20 (in increments of 5).

I used 10-fold cross-validation as the evaluation method to ensure robust performance measurement across the different hyperparameter combinations. For each iteration, I trained a decision tree model using the rpart function and evaluated its accuracy on the test dataset.

The optimal hyperparameters obtained were:

- cp = 0.001,
- minsplit = 5,
- maxdepth = 15.

These values resulted in the highest test accuracy of **0.7397**.

The fact that the accuracy remained the same (0.7397) when using both the optimized parameters from the grid search (*cp = 0.001, minsplit = 5, maxdepth = 15*) and the earlier control parameters (*cp = 0.001, minsplit = 10, maxdepth = 15*) suggests that the model's performance almost reached its ceiling with the current dataset and features.

3.6.2 Optimal Cp and Decision Tree Pruning

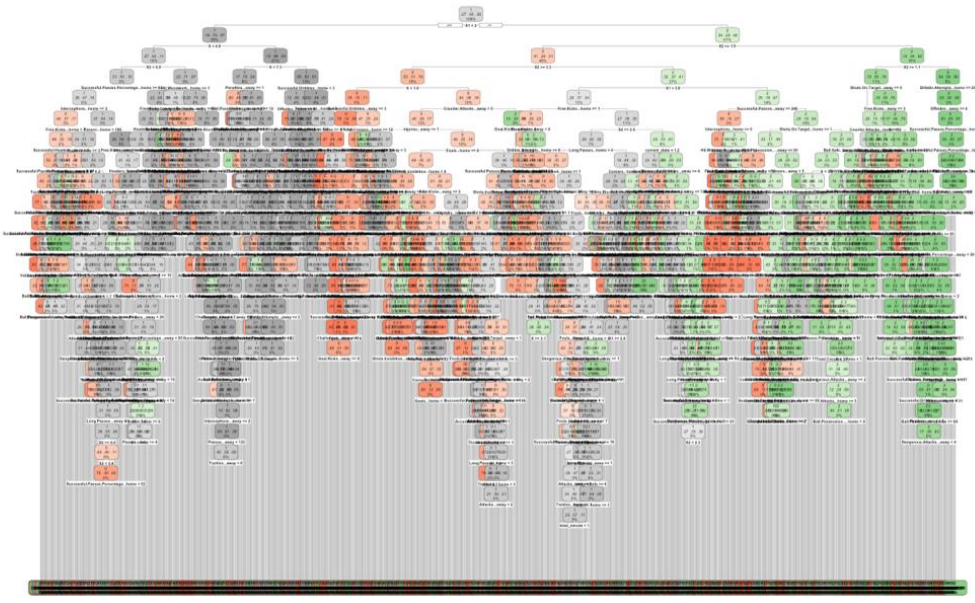
I trained a decision tree model on the full dataset without imposing any restrictions by initially setting cp = 0. This allowed the tree to grow to its maximum depth and complexity. Using the complexity table (cptable) generated by the model, I identified the *optimal*

complexity parameter (cp) as **2.233539e-05**, which minimized the cross-validation error (xerror). Once I determined this optimal cp value, I pruned the tree to remove unnecessary splits while retaining the most significant ones.

After pruning, I evaluated the refined model on the test set. The results showed a substantial improvement in accuracy, achieving **89.52%**, compared to the previous manually tuned model where the accuracy was only **73.97%**. The Kappa statistic also increased to **0.8392**, indicating a strong agreement between predicted and actual values.

However, while accuracy increased, interpretability decreased. This reduction in interpretability is due to the increased complexity of the tree after allowing it to grow deeper and incorporate more splits. Larger trees are harder to interpret because they contain more decision nodes and pathways, making it challenging to understand how the model arrives at its predictions. Thus, while accuracy improved, there was a trade-off in terms of the model's simplicity and ease of understanding.

I have provided the outputs I obtained below.



Over

```
    No Information Rate : 0.4302
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8392

 Mcnemar's Test P-Value : 0.1178

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
```

```
Sensitivity            0.8718   0.9173   0.8846
Specificity            0.9525   0.9327   0.9546
Pos Pred Value         0.8739   0.9114   0.8911
Neg Pred Value         0.9516   0.9373   0.9518
Prevalence             0.2742   0.4302   0.2955
Detection Rate         0.2391   0.3947   0.2614
Detection Prevalence   0.2736   0.4330   0.2934
Balanced Accuracy      0.9121   0.9250   0.9196
```

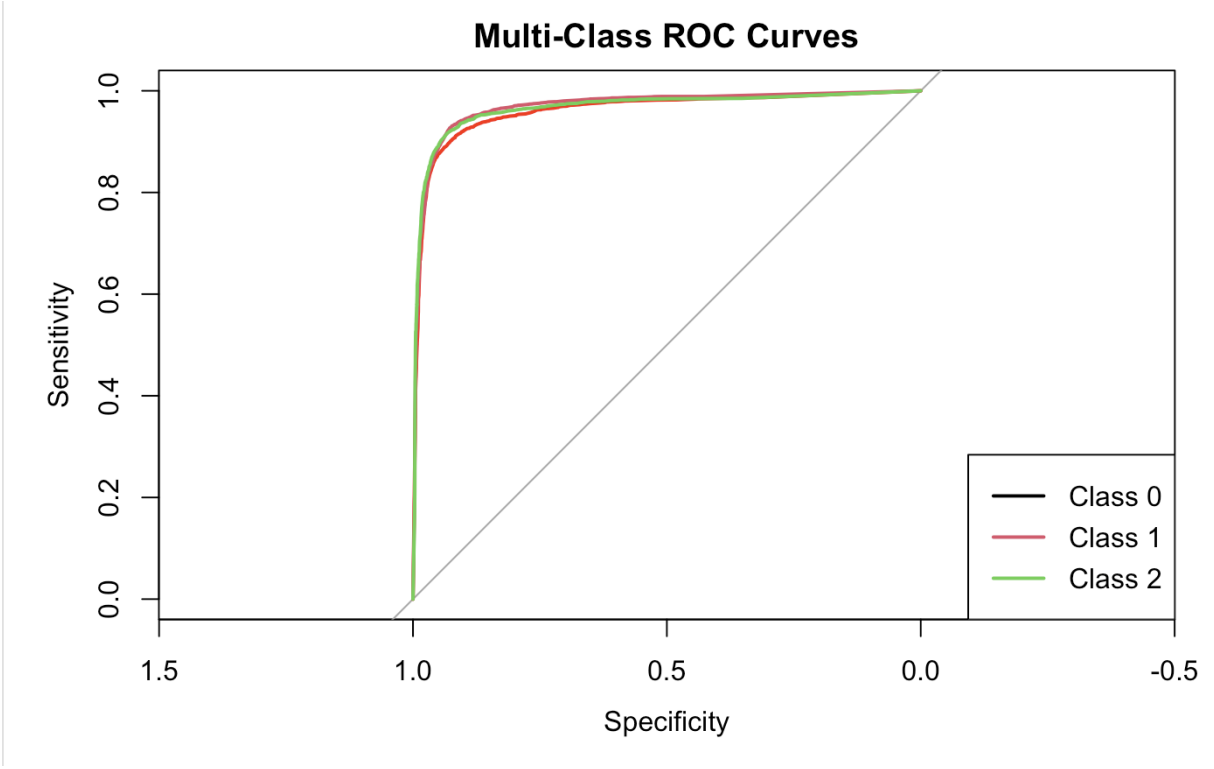
```
CART

39290 samples
   89 predictor
    3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 35362, 35361, 35361, 35361, 35361, 35361, ...
Resampling results:

  Accuracy   Kappa
  0.8845257  0.8228666

Tuning parameter 'cp' was held constant at a value of 2.233539e-05
```

The model I trained resulted in a decision tree with significant depth and complexity, which greatly reduced its **interpretability**. To assess whether this model might overfit, I compared the model's performance on the training and test datasets.

When I evaluated the model's training and test error rates, I found that the training error was 6.12%, while the test error was 10.48%. This resulted in an error gap of approximately 4.36%. Given this relatively small difference, I concluded that the model did not exhibit significant overfitting.

Here is the ROC curve and AUC values.

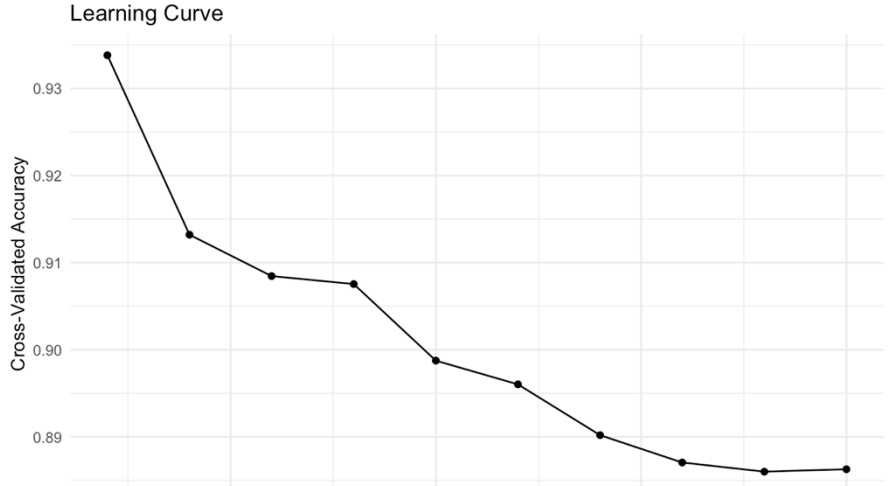
Ahmet Çeliker
2020402126




```
AUC VALUES:
     0         1         2
0.9607222 0.9690726 0.9660613
```

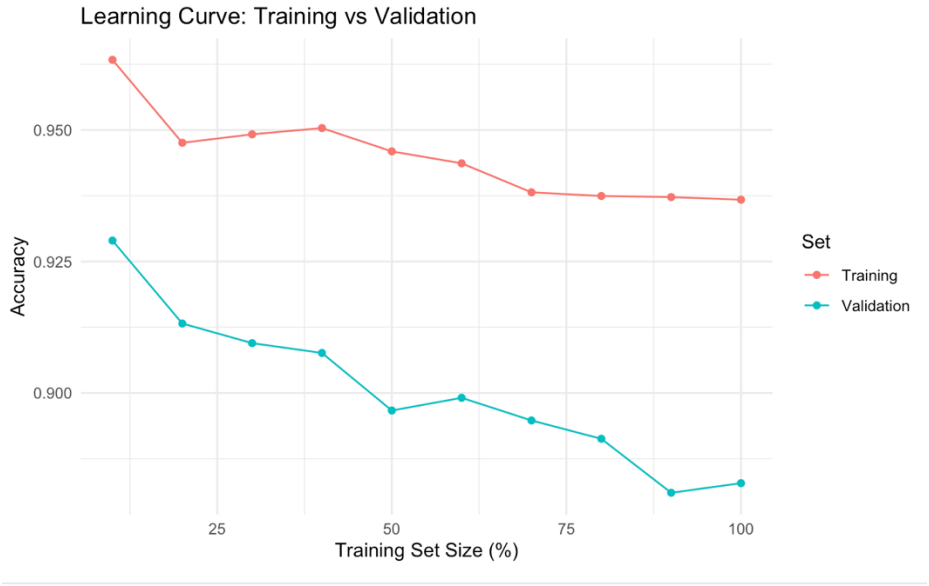
The ROC curve and AUC scores confirm that the model achieves near-optimal performance with minimal trade-offs between sensitivity and specificity.

I generated **learning curves** to better analyze the overfitting issue and evaluate how the model's performance changes with increasing training set size. I calculated the **cross-validated accuracy** for each training size and plotted a learning curve. This plot allowed me to observe how the model's performance improved (or plateaued) as more data became available.



Ahmet Çeliker
2020402126

I extended the analysis to compare **training accuracy** with **validation accuracy**. For each subset size, I trained the model and calculated the accuracy on the same subset (training accuracy) and on the cross-validated splits (validation accuracy). I stored these values in a dataframe and plotted them on the same graph, clearly distinguishing between training and validation accuracy.



The significant gap between training and validation accuracy in the second plot, combined with the consistent decrease in cross-validated accuracy in the first plot, indicates that the model exhibits **overfitting**.

Overall, while the model demonstrated strong performance, the learning curve confirmed that it still slightly overfitted the training data. I also lost interpretability of the model.

3.7 Boosting and Random Forest

In both Boosting and Random Forest, I observed that interpretability decreased significantly. This was primarily due to the increased complexity of these ensemble methods.

For Random Forest, the model combined numerous decision trees, each contributing to the final prediction. While individual trees were simple and interpretable, the aggregation of hundreds or even thousands of trees made it challenging to understand how the final predictions were derived. Tracking the interaction between trees and identifying the exact contribution of each feature became almost impossible.

In the case of Boosting, the process involved sequentially adding weak learners (such as small decision trees) to minimize errors made by previous iterations. This iterative optimization further increased the model's complexity, as each tree specifically focused on correcting the mistakes of the prior ones. As a result, explaining individual predictions or the overall decision-making process became quite difficult.

Ultimately, there was a clear trade-off: both Boosting and Random Forest improved accuracy at the cost of interpretability.

3.7.1 Boosting

I trained a boosting model using the XGBoost algorithm to predict the match results. First, I converted all factor variables in the train_data_selected dataset into numerical values, ensuring they could be processed by the XGBoost model. Each factor level was encoded as a numeric value.

Next, I separated the **target variable** (result) from the features and transformed the data into XGBoost's **DMatrix format**, which optimized performance and computation speed.

I defined the model parameters as follows:

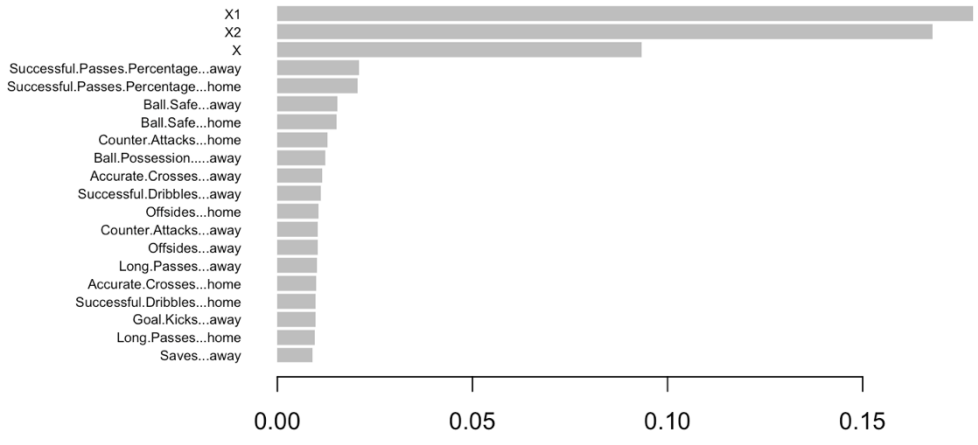
- **objective = "multi:softmax"** to indicate a multi-class classification problem with 3 classes (0, 1, 2).
- **num_class = 3** to specify the number of target classes.
- **eval_metric = "merror"** to use the misclassification error as the evaluation metric.
- **eta = 0.1** to set the learning rate and prevent overly large steps during training.
- **max_depth = 6** to limit the depth of each decision tree and reduce overfitting.
- **nthread = 2** to enable parallel computation using two threads.

I trained the model for **100 boosting rounds** and then used the model to generate predictions on the training data. By training this XGBoost model, I aimed to improve predictive performance compared to earlier decision tree models. Boosting's iterative nature allowed the model to focus on the errors of previous trees, making it a powerful approach for learning complex relationships in the data.

```
Below is the result of the model.

Overall Statistics

               Accuracy : 0.9178
                 95% CI : (0.9151, 0.9205)
    No Information Rate : 0.4302
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8731

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

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.8461   0.9631   0.9184
Specificity            0.9877   0.9166   0.9635
Pos Pred Value         0.9629   0.8971   0.9134
Neg Pred Value         0.9444   0.9705   0.9657
Prevalence             0.2742   0.4302   0.2955
Detection Rate         0.2320   0.4144   0.2714
Detection Prevalence   0.2410   0.4619   0.2972
Balanced Accuracy      0.9169   0.9398   0.9410
```

The overall performance of the **XGBoost model** demonstrated excellent results, with a high **accuracy** of **91.78%**. The results indicate that the XGBoost model significantly outperformed the previous decision tree models, both in accuracy and class-wise performance. The high sensitivity, specificity, and precision across all classes suggest the model effectively learned the underlying patterns in the data. However, there is a slight drop in sensitivity for **Class 0**, which could be further addressed with techniques like class weighting or additional feature engineering.

When I used the xgb.plot.importance() function with the parameter top_n = 20 and measure = "Gain", I visualized the top 20 features contributing the most to the XGBoost model's accuracy. The "Gain" metric specifically highlighted the features that played the most significant role in reducing prediction error.



The boosting model's feature importance confirms that betting odds (X1, X2, X) remain the strongest predictors, followed by key performance metrics like passing efficiency and ball safety.

3.7.2 Random Forest

Secondly I trained a **Random Forest** model to predict the result variable. My goal was to build a robust classification model and analyze the importance of each feature in predicting the match outcome. First, I used the randomForest function, specifying result ~ . to include all remaining predictors in the dataset after cleaning. By setting importance = TRUE, I ensured that the model calculated the importance of each feature, allowing me to identify which variables contributed the most to the predictions.

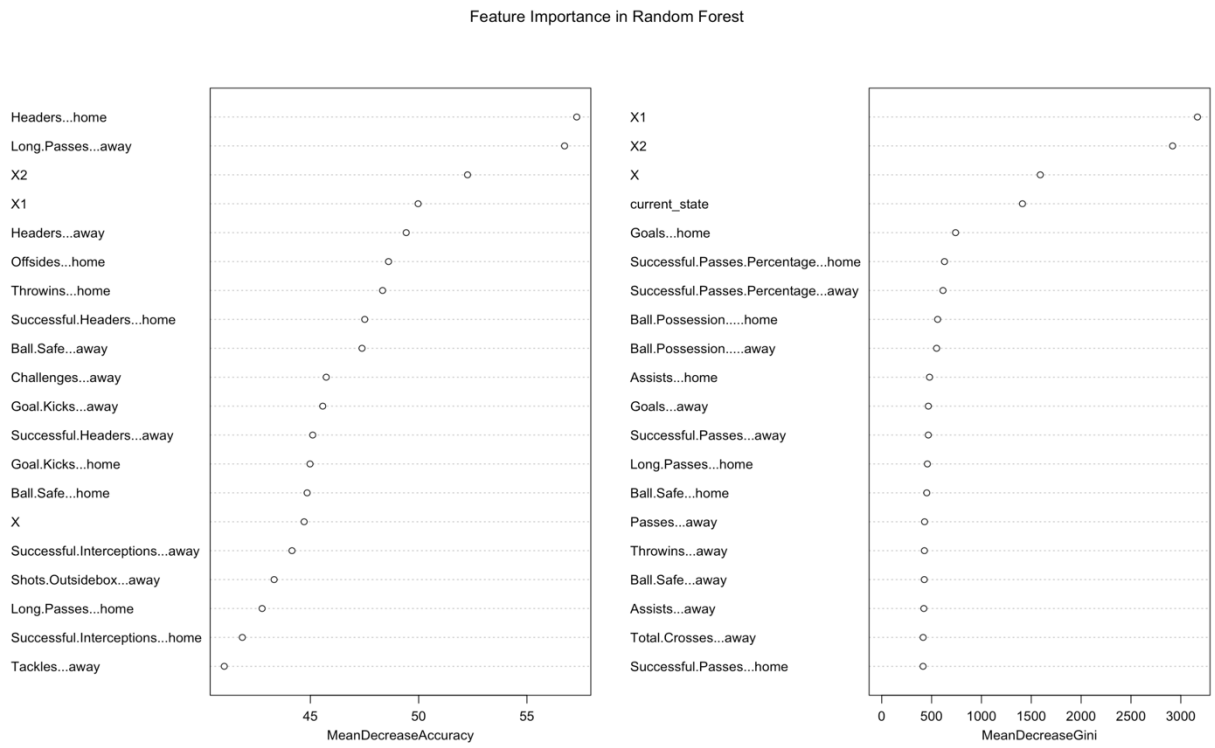
The output provided values like **Mean Decrease in Accuracy** and **Mean Decrease in Gini**. These metrics helped me understand how removing a particular feature would affect the model's performance and impurity reduction during tree splits. Features with higher values had a stronger influence on the outcome.

The output of the Random forest model is given below.

```
Call:
 randomForest(formula = result ~ ., data = cleaned_data_necessary,      importance
= TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 9

        OOB estimate of  error rate: 0.89%
Confusion matrix:
      0     1     2 class.error
0 15170   149    72 0.014359041
1    46 24020    82 0.005300646
2    23   130 16435 0.009223535
```

Ahmet Çeliker
2020402126

From the output of the **Random Forest** model, I observed that The **OOB error rate** was **0.89%**, indicating that the model performed exceptionally well on unseen data during training.



When I analyzed the feature importance plot from the Random Forest model, I noticed that **X1, X2, and X** were the most impactful features based on **Mean Decrease Gini**, indicating their strong influence in reducing node impurity and improving overall model performance. These features, likely representing betting odds, played a dominant role in determining the outcomes.

However, **Headers...home** and **Long.Passes...away** emerged as more impactful when looking at the **Mean Decrease Accuracy** metric.

As seen in both the Mean Decrease Accuracy and Mean Decrease Gini graphs, there is a significant gap following the top two features. After these two dominant features, the gap begins to **gradually decrease**, indicating a smoother distribution of importance across the remaining features.

After splitting the dataset into **70% training** and **30% testing** sets, I trained the **Random Forest model** and evaluated its performance on the test set. The resulting **accuracy** was **0.9993**, which indicates an extremely high predictive performance.

The result is given below.

```
Overall Statistics

               Accuracy : 0.9993
                 95% CI : (0.9988, 0.9996)
    No Information Rate : 0.4302
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9989

 Mcnemar's Test P-Value : 0.3251

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9989   0.9996   0.9992
Specificity            0.9999   0.9992   0.9997
Pos Pred Value         0.9998   0.9989   0.9994
Neg Pred Value         0.9996   0.9997   0.9997
Prevalence             0.2742   0.4302   0.2955
Detection Rate         0.2739   0.4301   0.2953
Detection Prevalence   0.2740   0.4305   0.2955
Balanced Accuracy      0.9994   0.9994   0.9995


Train Accuracy: 0.9367524
Test Accuracy: 0.8951713
```

From these values, I observed that the model performed **significantly better on the training set** compared to the test set. However, the difference is **not very large** (approximately 4.15%). This suggested that the model exhibited a **slight overfitting tendency**. If the gap between train and test accuracy had been **much larger** (e.g., 10-15%), I would have concluded that the model was clearly **overfitting**. In this case, the small difference indicated that the model still demonstrated **generalizable performance** on unseen data.

3.8 Comments

When I compared all the models in the context of this analysis, I found that ensemble learning methods, specifically **boosting** and **random forest**, delivered strong results but were **less favorable** in terms of balancing accuracy and overfitting. Despite their generally high performance, the models tended to slightly overfit, as observed in the accuracy gap between training and test datasets.

Ahmet Çeliker
2020402126

As the complexity increases with parameter tuning, accuracy improves, but interpretability decreases, creating a trade-off. I aimed to strike a balance between interpretability and accuracy.

Among the evaluated models, the **decision tree model** with the parameters:

- **cp = 0.001**,
- **minsplit = 5**,
- **maxdepth = 15**,

stood out as the **most optimal** solution. This model achieved a **high accuracy** while keeping **overfitting to a minimum**.

I analyzed the mean deviations between the model's predicted probabilities and the implied probabilities derived from betting odds to assess potential inefficiencies in the market. For home wins (X1), the mean deviation was 0.3763, indicating a notable discrepancy between the model's predictions and the market's implied probabilities.

For away wins (X2), the deviation was the highest at 0.5043. This large discrepancy highlighted a significant divergence between the model's assessment and the market's odds for away wins. This raised the possibility of market inefficiencies, which warranted further investigation.

The deviation for draws (X) was the smallest at 0.3031, indicating that the model's predictions for draws were more closely aligned with the implied probabilities. This suggested a level of consistency between the model and the market when evaluating the likelihood of draws.

When I built the model using the optimal cp, which achieved an accuracy of 0.8952, I observed the following mean deviations between the model's predictions and the implied probabilities: 0.4844 for home wins (X1), 0.5842 for away wins (X2), and 0.3939 for draws (X). Compared to the previous model, the deviations were noticeably larger across all outcomes.

To evaluate whether the deviations between the model's predictions and the implied probabilities are reasonable, I analyzed the underlying features and rules driving the model's decisions. The deviations observed, particularly for **away wins (X2)** and **draws (X)**, were consistent with the model's evaluation of the match data. These deviations likely arose

because the model considered features like **current_state, goals scored, and possession metrics**, which may not be fully reflected in the odds set by bookmakers.

For instance, the larger deviation for away wins (X2) could be explained by the model capturing patterns or trends in away team performance—such as high possession or dangerous attacks—that the market may have undervalued. Similarly, the moderate deviation for draws (X) might reflect the model's tendency to predict slightly higher draw probabilities in balanced match scenarios, which aligns with the data it analyzed but might not align with market odds.

3.8.1 The Effects of Features on the Deviation Between Predicted Probability and Implied Probability

I analyzed the matches with the highest deviations between the predicted probabilities and the implied probabilities for each outcome (X1, X2, X). I filtered the dataset to include only the rows corresponding to the highest-deviation matches. The features for these matches were extracted and analyzed to identify patterns or characteristics that might explain the large deviations. I observed that the features most correlated with deviations tended to be game-specific metrics, such as possession and crossing accuracy, or odds-derived probabilities like implied_prob_X2. This highlighted potential inefficiencies in the model, where certain features were either over- or underweighted in influencing predictions. Additionally, the strong correlation between implied probabilities and deviations (e.g., implied_prob_X2 for X1) suggested that the bookmaker's odds themselves could be driving some of the deviations. This raised questions about whether the odds were reflecting actual game dynamics or if the model was overly reliant on them.

REFERENCES

https://serdartafrali.medium.com/makine-ogrenmesi-cart-c318a49e0fdd
https://medium.com/@enes_flz/cart-classification-regression-tree-eab59258a126

https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680

https://www.geeksforgeeks.org/how-to-tune-a-decision-tree-in-hyperparameter-tuning/

https://towardsdatascience.com/multiclass-classification-evaluation-with-roc-curves-and-roc-auc-294fd4617e3a

https://betamatics.com

https://www2.it.uu.se/edu/course/homepage/projektTDB/ht15/project16/Project16_Report.pdf

https://www.digitalocean.com/community/tutorials/plot-roc-curve-r-programming