Hussam Taj

Math 495 - Mathematical Modeling

December 09, 2019

**NFL Big Data Bowl Project**

**Abstract**

For the Fall semester of 2019, my team and I decided to work on a Kaggle project. Kaggle is a platform where data scientists from all around the world come together to solve data science, machine learning, and predictive analytics problems. For our project, we decided to participate in the NFL Big Data Bowl project. The goal of this project was to develop a model to predict how many yards a NFL player will gain after receiving a handoff. Having little to no knowledge of machine learning, my team and I decided to take multiple machine learning approaches towards this problem. We tried different modelling techniques such as Random Forest, Adaptive Boosting, Gaussian Mixture, and Extreme Gradient Boosting. After doing some more research, we found out that many of these techniques are not the best approach for this specific problem; however, we managed to get some good predictions using a combination of the Gaussian Mixture Model and the Random Forest Model. Although some of the modeling techniques did not give us the desired results, we managed to vastly expand our knowledge about machine learning approaches and their applications in other areas of Data Science.

1. **Introduction**

Kaggle projects are extremely valuable for many reasons, such as providing real world problem solving experience. They also help job recruiters check our modelling ability by viewing our participation and performance, which can increase job opportunities. We are also able to seek the help of professionals by asking questions in the forum or discussion. This project was a great opportunity for my group because none of us have had any experience with machine learning. We took this on as a challenge and a great learning opportunity, since it could give us useful background information for next semester in the STAT 362 course, as well as help us understand how machine learning is used in the real world.

**1.1 The National Football League**

The National Football League, which is also known as NFL, is a mega corporation that governs over thirty-two individual teams (sportsrec). These teams are valued at billions of dollars, ranging from 2 to 5 billion dollars depending on each franchise. These teams start the season with the goal of becoming National Champions by winning the SuperBowl. In that pursuit, they compete weekly against each other, which draws thousands of fans to watch and support their favorite team. The team with the most wins earns the most amount of money, which can help attract the best players in the league to play for them and perform even better each year (statista).

To understand the project, we need to first understand American football. American football is a team sport played between two teams with each team having eleven players on the field. The field is a 100 yard long rectangular shape and it has a goal post at each end (sportsrec). The offensive team has the ball and their goal is to take the ball to the other side of the field. They can achieve that by running with the ball or passing it to another team member. In four plays, which are also known as “downs”, the offensive team must advance at least ten yards; if they fail to achieve that, they need to turn the ball over to the other team. On the other hand, if they succeed to achieve that, they get another set of 4 plays to keep advancing. If they advance the ball to the end zone for a touchdown, they score points. In the meantime, the defense team, which does not have the possession of the ball, tries to stop the offensive team from advancing by tackling their players, especially the one who is holding the ball. At the end, the team with the most points wins the game (sportsrec).

**1.2 Project Goals**

The NFL Big Data Bowl project was a very interesting project for us because sports have always been exciting to us. However, I was not familiar with American Football, so I had to learn about the sport on top of learning about machine learning techniques that could be applied to it. This project will be very beneficial for the sport, as it will help teams, media, and fans better understand the skills of the players and the strategies used by the coaches. It will also assist the teams to assess their coach, the opposing team, the ball carrier, and his teammates in making necessary adjustments (Kaggle). We were provided data from the 2017-2018 NFL season and it was a time-series API, which means it was recorded over a period of time in equal intervals. We were supposed to train our model on this given data so that the evaluation system can test it on data from the games happening between November 28, 2019 and December 29, 2019. On January 6, 2020 the results of the competition will be published.

1. **Data**

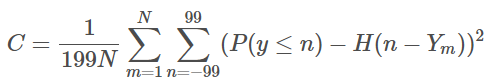
The data provided has 49 different variables such as offense formation, defenders in the box, the position of the player, temperature, location, stadium, etc. The data was collected from 23,171 unique plays, and since each play has multiple observations within, this gave us a total of 509,763 observations. The first step to approach a problem like this is to do an exploratory data analysis and clean up the data. The data we had was in a good format but there was still a lot of organization and tidiness required to get it to the right format. Then we could perform some feature engineering and exploratory data analysis to have it ready for our modelling steps. There were some variables in the data set, particularly the categorical variables, that required some cleaning up. The “OffenseFormation” variable for example, had some blanks that needed to be changed to “Unknown”, and “PlayerHeight” was converted to inches. The “Stadium” variable had multiple names listed to represent the same stadium (i.e. FirstEnergy Stadium was listed as “FirstEnergy Stadium”, “First Energy Stadium”, “FirstEnergy” and “FirstEnergyStadium”). They needed to be converted into a single value representing one stadium. Also, the New York Jets and Giants play in the same stadium, so they were being accounted for twice. “Turf”, “GameWether”, “Location”, “StadiumType”, “WindDirection”, “PossessionTeam” and some more variables were cleaned in a similar way.

For the feature engineering, we had to create different indicator variables to identify different situations during a play, but it was specific to the model being used. Some of those examples include creating new variable, such as *isRusher* to identify who is the rusher, *PlayerTeam* to indicate if the player is on the offense or defense team, *TeamMargin* to indicate whether the player’s team is in the lead or not, and i*s\_TwoRBSet* to indicate whether the offensive set has more than one running back or not.

Finally, for exploratory data analysis, we started by analyzing the relationship between our variable of interest—*Yards*—and other variables in our training set. We found some good explanation for our data set and the relationship between different variables. Some of the most interesting results we found were that there was no difference between the yards gained and playing surfaces, which contradicted our original hypothesis that artificial turf would be less susceptible to weather conditions and would help teams gain more yards. We also built a simple linear regression to get an idea of the most influential variables in our data set. We started with having *Yards* as our response variable and all other variables are our predictor variables. We used stepwise AIC as our variable selection criteria to select the variables that prove to be the most influential for our model and eliminate the variables that do not. We found that *GameClock, VisitorScoreBeforePlay, DefendersInTheBox, and Distance* were all influential to the model. *CollegeName*, *PlayerNumber*, and some other descriptive variables did not prove to be influential as they do not provide information on the number of yards gained by the runner. After identifying influential variables, we had a good idea of our data set and the relationship between different variables, so the nextstep was to look into different machine learning techniques to solve our problem.

1. **Submission system**

One of the first challenges we faced was understanding the submission system on Kaggle. It was a test because our response was a single yard value, but the metric used to score the models was a Continuous Ranked Probability Score (CPRS), which accepts a cumulative probability distribution. The function used is given below:



Here, N represents the number of plays in the test set, Y is the yards gained in the rushing play, and P is the predicted distribution, H is the Heaviside step function where H(x) = 1 when x > 0 and H(x) of every other value returns zero. Although R was one of languages provided by the NFL administration as an option toto solve this problem, the submission system was accessible through Python environment. Fortunately, the Kaggle community created an R script that used the Python interface package to access necessary Python environment. This way, we were able to use our preferred programming language to do this project.

Just to test the submission system format, we created a distribution of rushing yards for each rusher by calculating the mean and standard deviation. There was no machine learning involved with this process; it was a simple normal distribution with player’s historical mean and standard deviation over 199 yards interval. To our surprise, this model scored much higher than what we had expected and gave us a very good initial score to start with. This score remained as our best model for a significant portion of the competition. Having understood the submission system, we started to further explore machine learning approaches and methods.

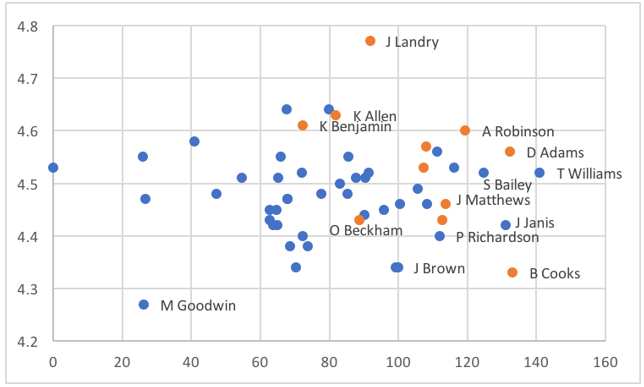
1. **Methods**

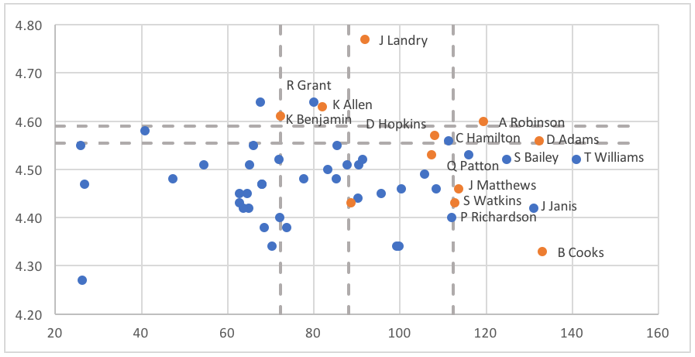
After getting data in the right format and understanding the submission system, we started researching different machine learning approaches for this problem. We looked into many different modelling techniques that could possibly help us achieve our desired solution and our goal of getting the most accurate prediction. But as a group, we did not have enough time to fully focus on each technique. So, each of us decided to focus on a certain modelling technique while keeping all other group members updated.

**4.1 Adaptive Boosting**

One of the machine learning techniques we tried for this problem was Adaptive Boosting. Adaptive Boosting, also known as AdaBoosting, is the first the boosting algorithm proposed by the two data scientists Freund and Schapire in 1996. This algorithm is used for classification problems as its objective is to turn weak classifiers, which are also called learners, into strong classifiers. The basic idea of the algorithm is that for any learner with accuracy higher than 50%, the weight is positive, and for any learner with accuracy lower than 50%, the weight is negative. The weight depends on its accuracy. This way, we can combine its prediction by just flipping the sign. So, even if a learner performed worse than just random guessing, it still contributes to the prediction (towardsdatascience). Using this algorithm, we can compile many weak learners (less accurate predictions) into one strong learner (high accuracy predictions).

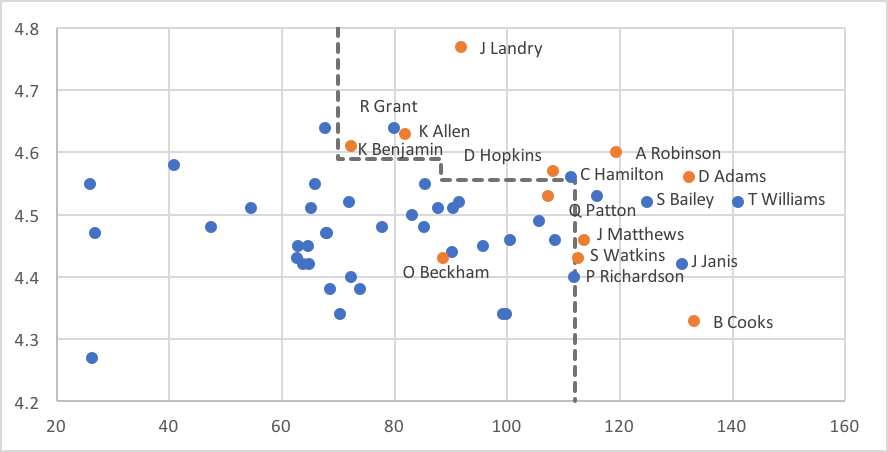
For this project, our goal was to predict how many yards a player will gain after receiving a handoff. Using the idea of adaptive boosting algorithm, we combined multiple weak yard predictions into a single strong yard prediction. We created a binary indicator variable that was used as a response for each single yard value. We used the indicator value to be 1 if the yards gained were more than five; else the indicator value was 0. By treating our problem as a classification problem, we assigned the indicator 1s for orange and 0s for blue, and using R library Adabag we were able to produce some iterations as shown below. We have yards per game on the x-axis and 40-yard dash on the y-axis.



**Figure 1: Original** 

**Figure 2: 1st iteration**

Figure 1 shows the original classification and Figure 2 shows the 1st iteration. In the first iteration, there are five splits done with the grey dotted line. After the first iteration, the model splits depending on the weight, and as there are more 1s on right side of the graph, it showcases more 1s on the right of the vertical lines. After many iterations, the model takes weighted average of the distribution.



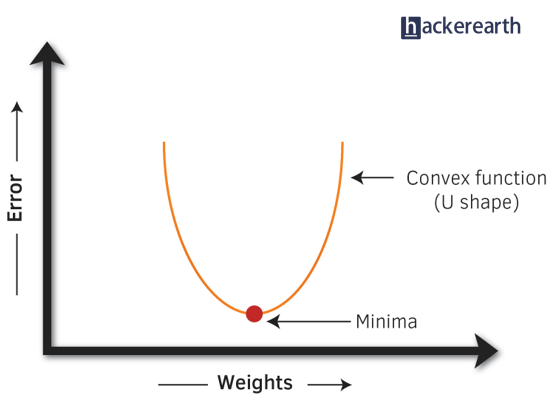
**Figure 3: Current model**

In figure 3, the current model was able to classify the 1s and 0s; there are some points that are still not classified, but some errors and inaccuracies are inevitable. The error rate was 0.947-0.683, which was very high. This is because there are many confounding factors such as not all variables were accounted for in our classification that can be misinterpreted by the model. Due to high inaccuracy, we decided to look into more machine learning approaches to see if we could achieve an accurate yard prediction with a low error rate.

**4.2 Extreme Gradient Boosting**

Extreme Gradient Boosting, also known as XGBoost, is one of the most famous and liked machine learning algorithms at Kaggle (kaggle). Many teams have had a lot of success in winning the competitions because it can be used for classification and regression problems. XGBoost was created by Tianqi Chen, a PhD Student at the University of Washington in June 2016. This model uses gradient boosting framework at core but performs better than the rest of boosting models. This is because it implements gradient boosting decision trees which are designed for speed and performance. It is unique because it uses a more regularized model construction which prevents it from over-fitting, resulting in better performance. We needed to understand XGBoost and its functionality before we could apply it our problem.

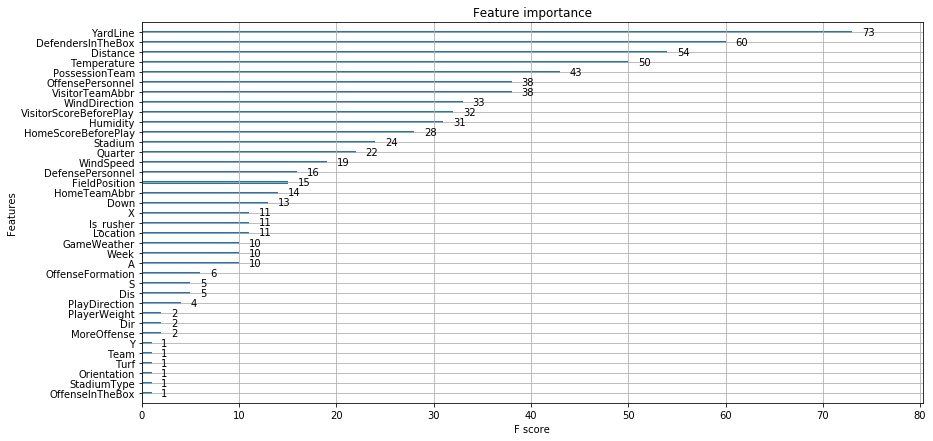
Boosting algorithms are typically used to convert weak learners into strong learners. This is achieved through an ensemble technique where new models are added to correct the errors made by existing models. Models are added until no further improvements can be made. Adaptive boosting is similar to gradient boosting; the only difference is that gradient boosting generates learners during the learning process. The second learner is built to predict the loss after the step. The sample distribution is not modified, rather the weak learners are trained on the remaining error of the strong learners. The weight of each learner depends on the prediction accuracy of the sample. Gradient boosting uses the idea of gradient decesnt in which the partial derivative of the loss function is calculated to find the minima with respect to the prediction as shown in figure 4 below. In simple words, gradient descent optimizes the loss function by tuning different values of coefficient to minimize the error. Tuning is a major factor in extreme gradient boosting as it helps achieve the lowest error value and the highest prediction value. Using xgboost without tuning parameters is like driving a car without shifting gears (hackerearth).



**Figure 4**

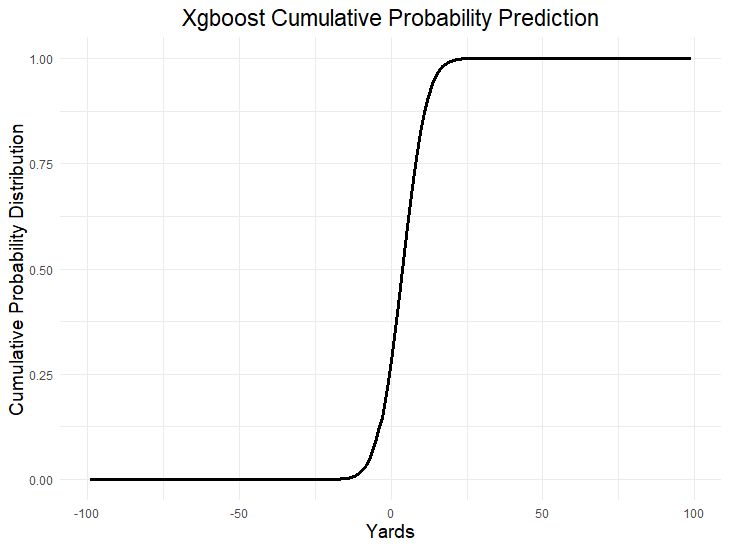
As tuning plays a major role in XGBoost, there are different parameters that play a significant role in the model’s performance. The parameters can be divided into three categories: general, booster, and learning task parameters. The general parameters control the booster type that drives overall functioning in the model, the booster parameter controls the selected booster and sets and evaluates the learning process of the booster from the given data (hackerearth).

For our problem, we tried to use a regression approach with extreme gradient boosting. We used the R package “xgboost” to run the algorithm on our data. The booster we used was gblinear, which builds a generalized linear model and optimizes it using regularization, tree regression, and gradient descent. Regression tree uses the same idea as decision tree, which is used for all the tree based algorithms such as random forest. The decision tree contains one score in each leaf value and the prediction is the sum of scores predicted by each tree (homes.cs.washington.edu). Using this algorithm, we had to use cross validation and split our data 90 to 10 using random sampling. 90% of the data was used for training set and 10% for testing set. After splitting the data, we used indicator variable for a single yard value to be our response variable similar to how we used the same idea in adaptive boosting in earlier section After filtering the data to just the rushers by having playID of players be equal to rusherID, we ran our model. We used feature importance as our variable selection technique and got the graph given below:



**Figure 5**

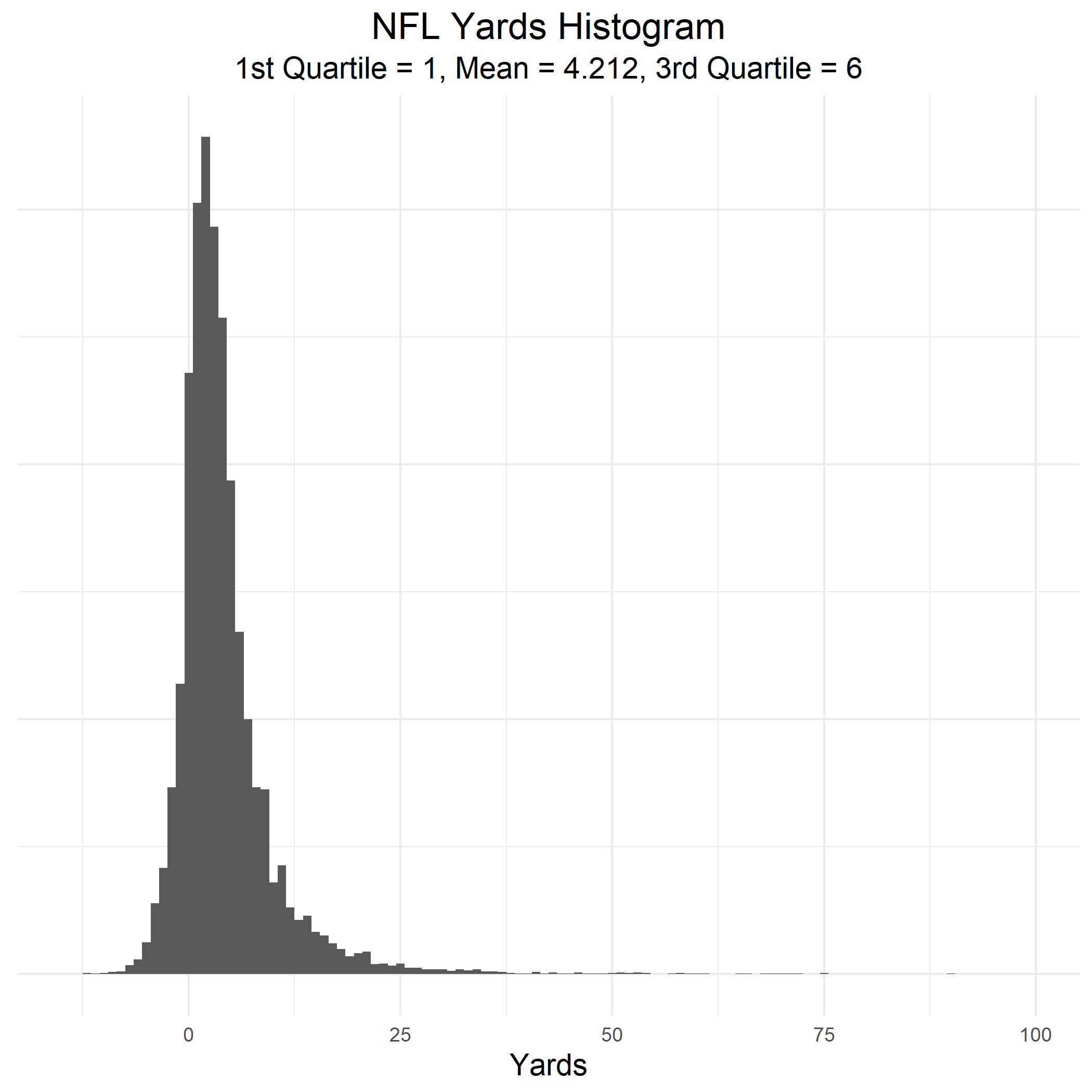
The F-score implies the relative contribution of the corresponding feature to the model calculated by taking each feature’s contribution for each tree in the model. The graph shows that *Yardline, DefenderInTheBox, Distance, Temperature,* and *PossessionTeam,* and *OffensePersonnel* are the 5 most important features, while there are many more listed as well. Using different important variables, we were able to get a prediction for a single yard value. After getting a single yard prediction, we created a loop to get the number of yards on each individual play. Using these predictions, we created a cumulative density function which is shown below.



The distribution looked normal and centered around zero, but our root square mean error was 5.98, which suggests that our accuracy rate is very low. There are many reasons for low accuracy. The first one is that our variable selection method could be misinterpreted, as feature importance trains on all the data se;, if variables are missing some values, it gives them a higher F-score without comparing the proportion size. Secondly, to get a better prediction, we needed to tune the model, which is a very advanced method of XGBoost algorithm that we did not have expertise to implement. Due to these reasons, we decided to focus on other machine learning approaches that could give us a higher prediction.

**4.3 Gaussian Mixture Model**

One of our successful approaches to this problem was using the Gaussian Mixture model. The idea of using a Gaussian Mixture model came when analyzing the distribution of the NFL yards shown below in Figure 6.

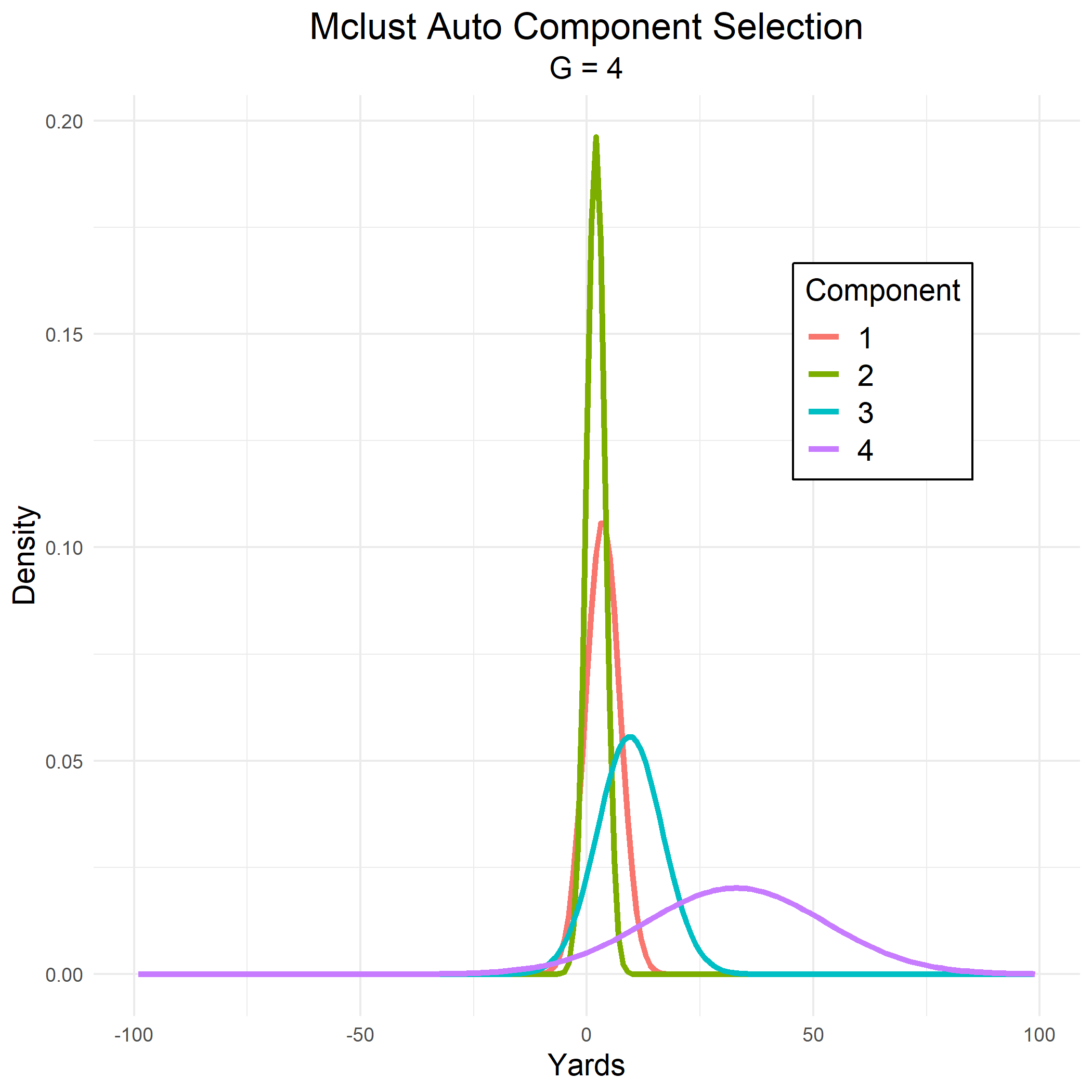


**Figure 6**

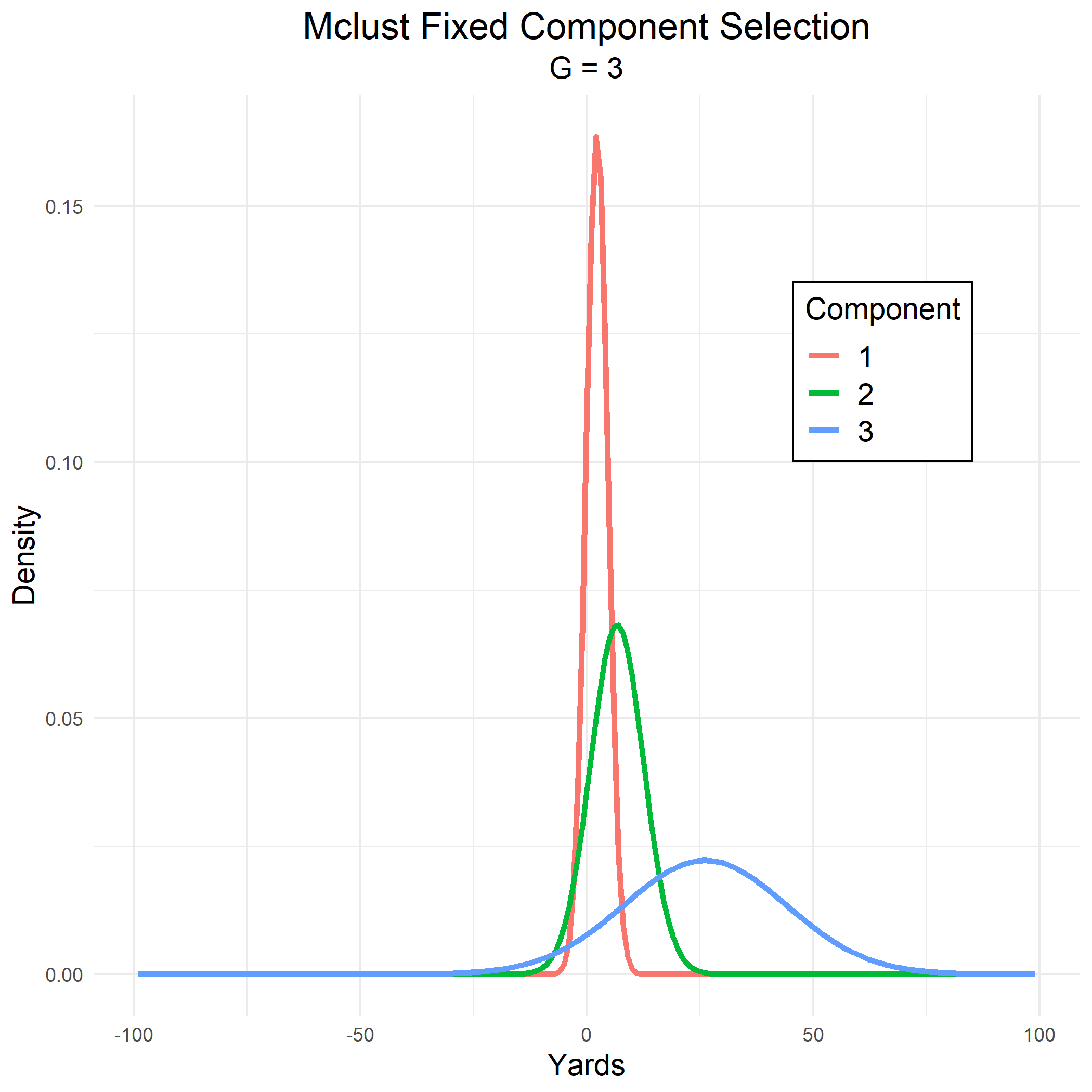
The minor jumps between 10 to 20 yards seemed interesting to us as we think it was a right skewed Gaussian distribution or a combination of different distributions. 54% of rushing plays were contained between zero to five yards and 70% of rushes in the training set were contained between zero to ten yards. This fits the idea that usually the rushers are not able to break through the defense and they cannot pass 10 yards or more. However, when they do get a chance to pass the defense, they’re able to gain more yards. This made us wonder whether these jumps were due to these types of plays.

To further look into this theory, we started to research Gaussian Mixture models. We used the R package mclust which defines itself as Gaussian Finite Mixture models. This package could be used to identify if there is a combination of multiple distributions where the plays could be normally distributed with a mean of few yards and a small standard deviation, and the plays where a rusher would get passed the defense and the distribution would have a higher mean and standard deviation. This could potentially be turned into a classification problem and we could implement some machine learning algorithms which could help us predict which distribution each play belongs to.

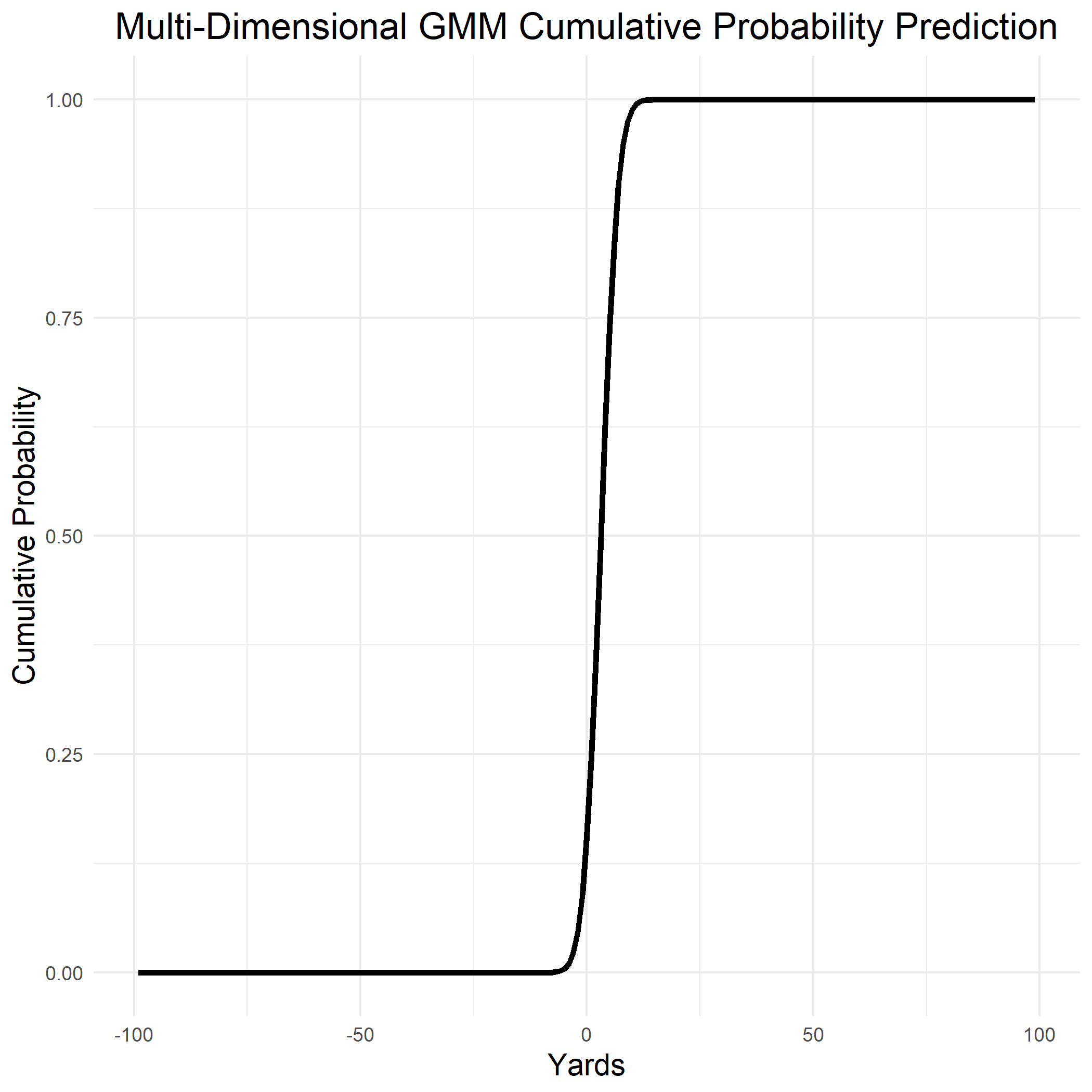
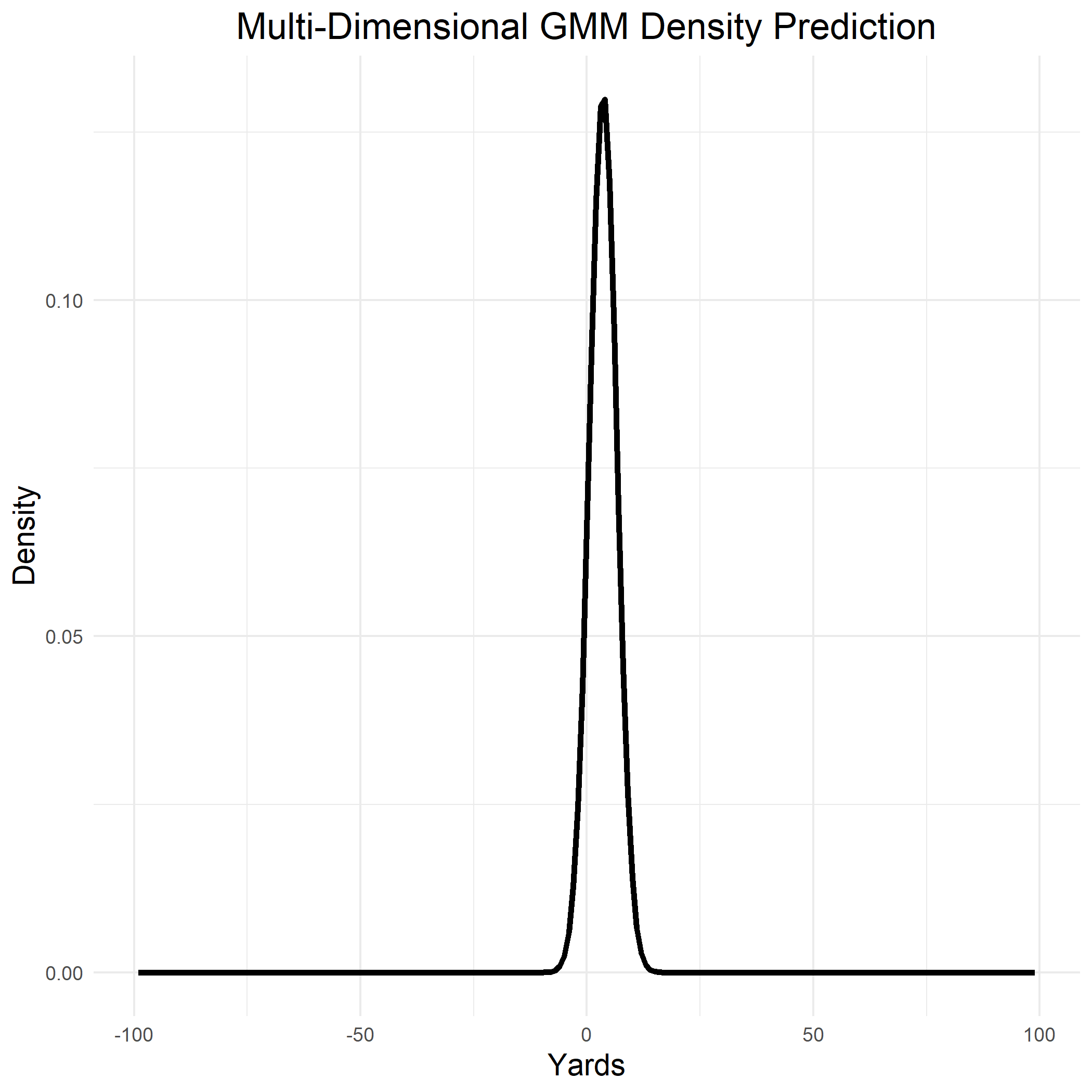
Using the package mclust, the model ran on the training data and calculated the best four components . Those components had a distributions each which are represented in the graph below with their respective parameters. It shows the density of each distribution and their distance from each other.



The four components did not seem differ drastically from our expectations, but the center of distribution was not where we expected. After noticing that component 1 and 2 had similar means and distributions, we decided to look into combining both components and analyze their effects on the distribution; however, it did not result in any improvements, as shown below.



After some basic analysis of trying to find a combination of different distributions, we decided to look into other directions using Gaussian Mixture model to get a solution for this problem. We were able to insert different variables into the mclust function to create multiple dimensions in the estimators. We used *yard line, down, distance, defenders in the box, and total seconds remaining* to train the model on yards and get an estimate of the density for different yard values. As our prediction frame accepted 199 values (from -99 to 99), it would repeat every variable 199 times during the process, which allowed us to hold all variables constant and predict the density of each yard value, giving us a complete distribution as shown below. Although this method seemed promising when we used some variables, as we added more variables, it did not scale well and did not give us a good score.



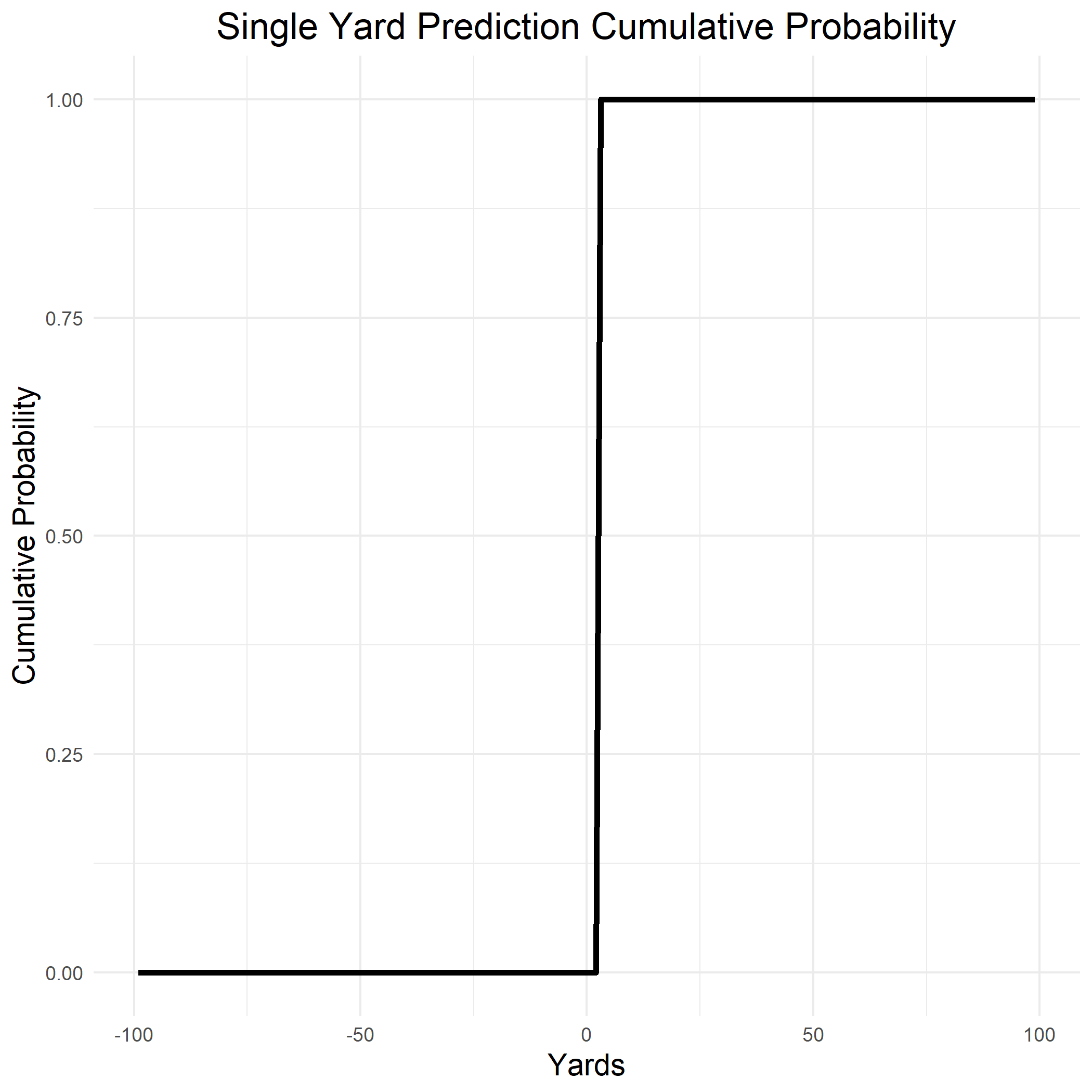
Since all the methods were not helping us get a good prediction with high accuracy, we tried to combine two of the methods we were using in hopes of getting a better score. We found some good results by combining Gaussian Mixture with random forest model.

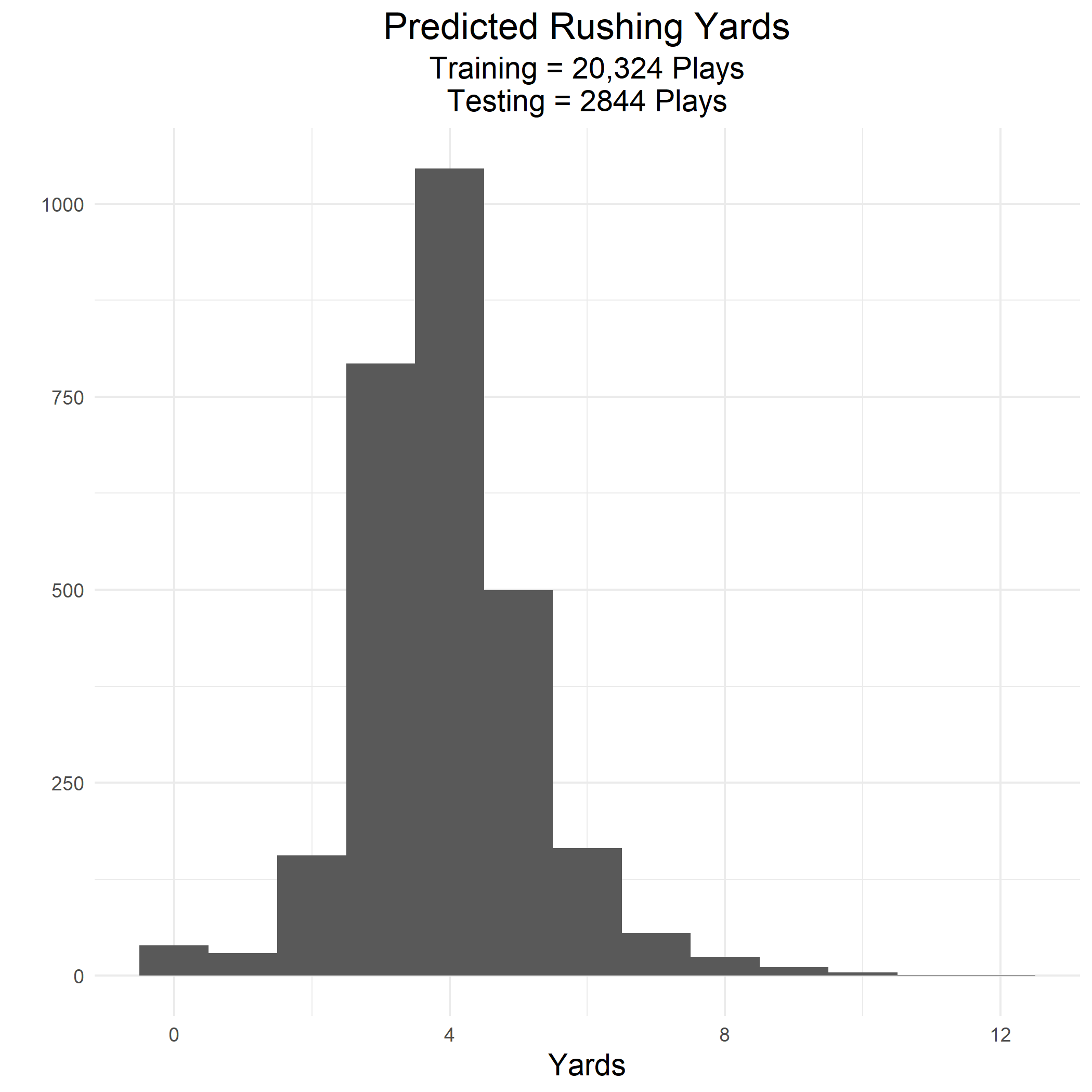
**4.4 Random Forest**

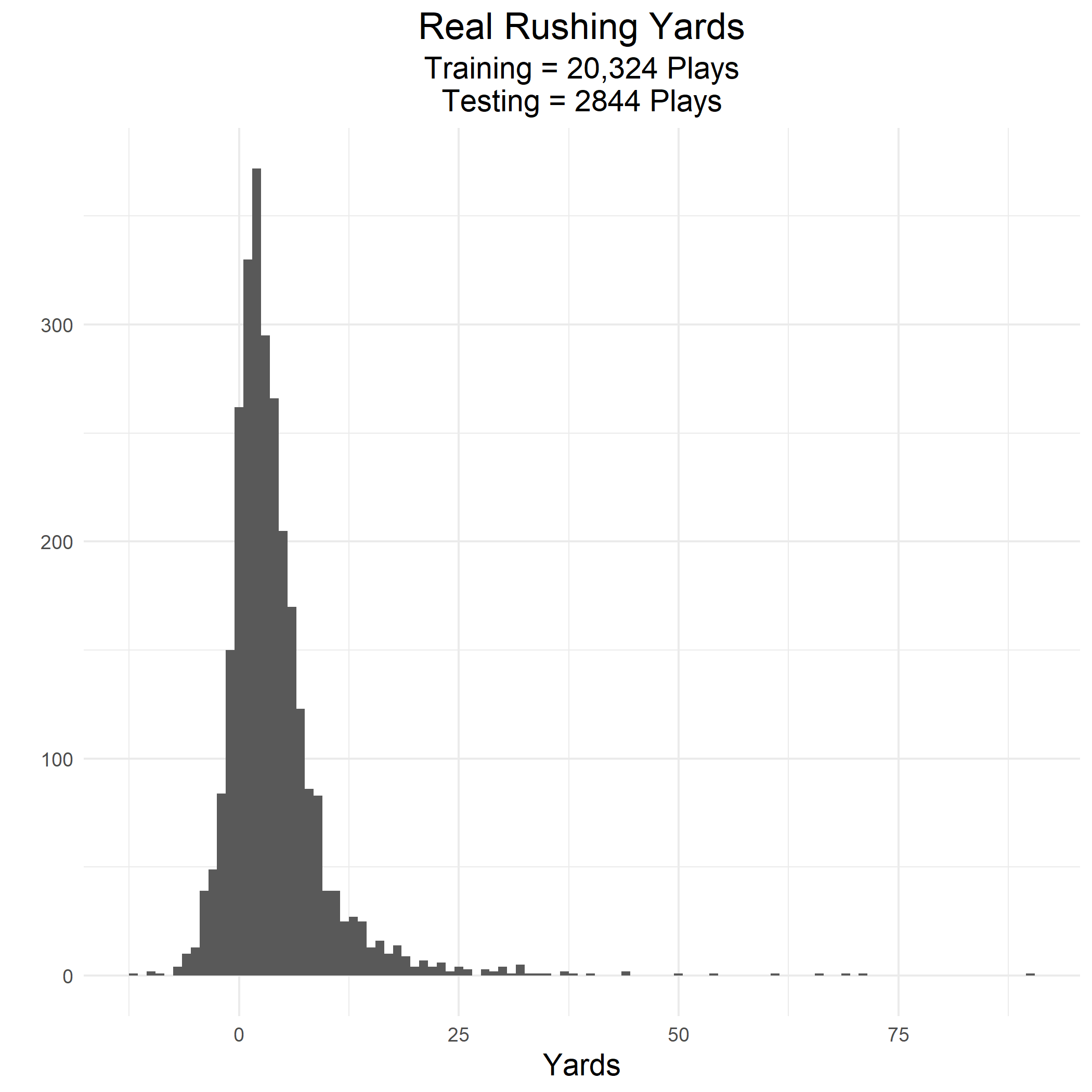
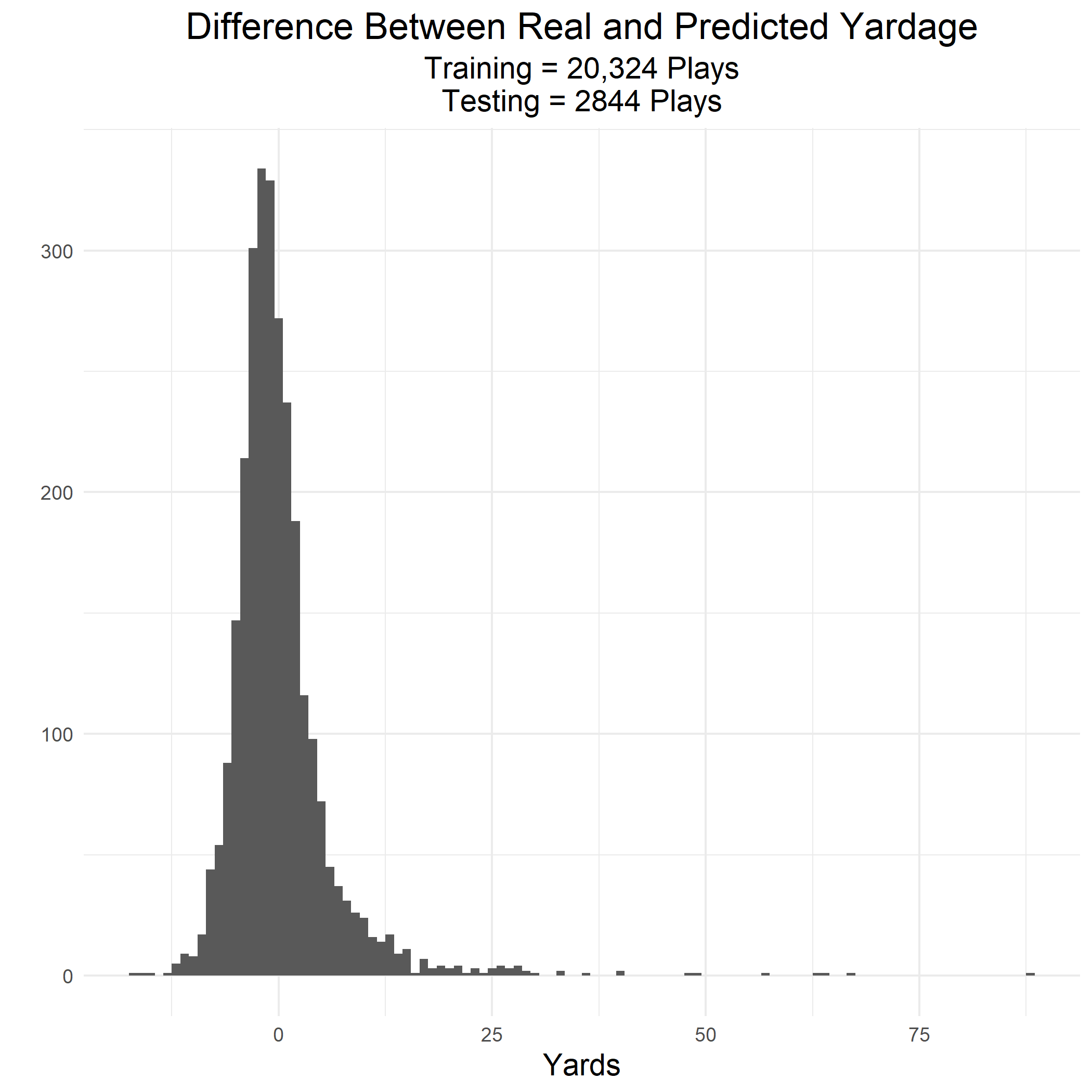
Another machine learning technique we used is called random forest algorithm. It is a very powerful algorithm which works by creating multiple decision trees and then combining the output by each decision tree. It is best applicable with classification problems because it uses decision trees and the concept of information gained at every node. At each node, classification is used to check if information is gained. After many iterations the classification where most information was gained is recorded. It keeps following the process until no further information is gained. Decision trees are very simple and easy to understand, and they have a very low predicting power—which is why they are called weak learners. Random forest uses multiple decision trees and combines their output to produce one output. This output does not take into account all the data points or decision trees; it actually randomly samples the data points and variables, which removes the bias that a decision tree model might introduce in the system (r-bloggers).

For our project, we used cross-validation to split our data into training and testing sets. 90% of the data was split for the training set and 10% for the testing set. We then used Randomforest R command to get a prediction for a single yard value. After getting the prediction, we created a loop to get the number of yards on each individual play. The first model did not get us a good score in the submission system.

After not having a significant success with the Random Forest and Gaussian Mixture model, we were able to apply a random forest to our Gaussian outcome, which greatly improved our accuracy. As the random forest predicted a single yard value and not a distribution, we were able to obtain a cumulative probability distribution which is shown below:



This model looked the most promising compared to rest of the models, and after testing the accuracy of our model with the prediction of a single yard value, we found some hopeful results which are represented through graphs below:



The figures above demonstrate that our model does not have a wide range of values as the real rushing yards have, which is expected with a model as simple as this one. We have also observed that our model has not been able to get good accuracy for extreme values, or predict zeros and negative yards. This was concerning because 11% of rushing plays ended up in yards lost. Although many Kaggle community members were able to take these factors into account, we did not have the expertise and knowledge to focus on them. Regardless of this issue, this model seemed to be the most reliable in getting us the desired solution. We believed that this model would give us a high score, but unfortunately, it scored poorly. We were surprised that our best model scored much worse than our first submission. After some reflection, we hypothesized that the poor score was due to our submission format. We realized that as our initial model was a normal distribution, we were jumping from zero to one at the predicted yard. To solve this issue, we decided to use the predicted value for the random forest as the mean for the normal distribution and the standard deviation would be created by the rusher’s historical data.

This method improved the model’s score and brought it very close to the initial submission. As our model was centered around the same yard value as the original random forest model and the distribution was the standard deviation of rusher, we considered whether standard deviation had a role in the submission score. After testing different values for the standard deviation, we found that having high and low standard deviation values were not beneficial for the model, but having 5.248 as our mean (which was mean of rusher yards) gave good scores. Keeping this in mind, we replaced all the extreme standard deviation values with the mean of rusher and it gave us the best score for the competition.

**5. Conclusion and Future Work**

This project has been a great learning experience for us, and there are many big takeaways from it. First, we started with no knowledge of machine learning and after doing intense research on many different machine learning approaches, we learned the theory as well as its implementation on real world problems. We gained a lot of real world experiences such as solving a problem with no set framework to follow and understanding that not all problem solving techniques are always applicable. We also gained experience with working with “dirty” data and how important and complex data cleaning is. We also learned how to be effective team members by helping each other get a solution or at least the right approach. Although we are proud of our progress and achievements in this project, we could ve gotten a better score for our models if we had machine learning background and knowledge. Nevertheless, this was a great way to step into the machine learning world and we are excited to learn more about this field in the future.

Bibliography

“NFL Big Data Bowl.” *Kaggle*, www.kaggle.com/c/nfl-big-data-bowl-2020/overview.

Bass, Alan. “How to Take Statistics at a Football Game.” *SportsRec*, 15 Oct. 2019, www.sportsrec.com/take-statistics-football-game-4441153.html.

“Beginners Tutorial on XGBoost and Parameter Tuning in R Tutorials & Notes: Machine Learning.” *HackerEarth*, www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/.

Brownlee, Jason. “A Gentle Introduction to XGBoost for Applied Machine Learning.” *Machine Learning Mastery*, 21 Aug. 2019, machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

Fuller, Steve. “Topic: National Football League (NFL).” *Www.statista.com*, www.statista.com/topics/963/national-football-league/.

“How to Implement Random Forests in R.” *R*, 9 Jan. 2018, www.r-bloggers.com/how-to-implement-random-forests-in-r/.

jaseziv83. “Comprehensive Cleaning and EDA of ALL Variables.” *Kaggle*, Kaggle, 15 Oct. 2019, www.kaggle.com/jaseziv83/comprehensive-cleaning-and-eda-of-all-variables.

Luca Scrucca, Michael Fop, T. Brendan Murphy and Adrian E. Raftery. *Https://Journal.r-Project.org/Archive/2016/RJ-2016-021/RJ-2016-021.Pdf*.

Nishida, Kan. “Introduction to Extreme Gradient Boosting in Exploratory.” *Medium*, Learn Data Science, 21 Mar. 2017, blog.exploratory.io/introduction-to-extreme-gradient-boosting-in-exploratory-7bbec554ac7.

Nishida, Kan. “Introduction to Extreme Gradient Boosting in Exploratory.” *Medium*, Learn Data Science, 21 Mar. 2017, blog.exploratory.io/introduction-to-extreme-gradient-boosting-in-exploratory-7bbec554ac7.

“Random Forests.” *Random Forests · UC Business Analytics R Programming Guide*, uc-r.github.io/random\_forests.

SauceCat. “Boosting Algorithm: AdaBoost.” *Medium*, Towards Data Science, 30 Apr. 2017, towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c.