# Predicting the Number of Rushing Yards in a Given NFL Running Play

## Grant Nolasco

### Abstract

Recently, there has been an increased use in player tracking and analytics in the National Football League during the past decade, which leads to use a huge question that might prove beneficial to coaches and general managers: is machine learning able to predict the number of yards in a given rushing play and what features contribute to a successful running play? Using player tracking data provided by NFL Next Gen Stats from 31,007 running plays during the 2017-2019 seasons, random forest and convolutional neural network algorithms were used to train a model that could potentially be useful in answering those questions. After comparing each model, neural networks were found to be a better algorithm in predicting the number of rushing yards in a given play.

### Introduction

One of the biggest sports developments during the 21st century has been the use of analytics and machine learning to create informed decisions for coaches and general managers to help improve their respective teams. For example, Billy Beane and Theo Epstein, who are credited with turning around the luck of their respective franchises by bringing postseason success after years of failures, relied on data to objectively evaluate players and create analytical-based strategies and decisions. Even the National Football League has taken strides to provide necessary data to teams to help them understand player/team performance and trends. In

2014, NFL started tracking players on the football field using RFID tags on their shoulder pads, which provides analysts and data scientists a much deeper understanding of football as they have information of a player's movement, location, direction, and more at every second of the game [1]. With player tracking data, potential topics that have been difficult to understand like quantifying defensive performance on passing plays outside of the traditional metrics (e.g interceptions) could finally be researched and answered. One of those difficult topics is understanding the context behind a successful running play and the ability to predict the number of rushing yards at the time of handoff, which would provide meaningful insights to NFL coaching staffs. The purpose of this study is to understand which features and algorithms would correctly predict the number of rushing yards in a given play.

## Methods

### Data Information

The dataset comes from NFL Next Gen Stats, which captures real time player tracking data such as their speed, acceleration, direction at every second of an NFL game. For this Kaggle competition, which was hosted by the NFL in 2019, the training dataset includes all 31,007 running plays starting from the beginning of 2017 season to Week 12 of 2019 season [2]. There are 682,154 rows (each play has one row containing information about each player at the time of handoff ) and 49 columns, which includes their position and speed at the time of handoff and other information about the play and game like offense/defense formation that was used during the play and the weather during the game. The variable *Yards* is the response variable that we're interested in predicting.

**Rules**

Before submitting the code for evaluation, the NFL had certain rules for the Kaggle competition that I had to consider before I could start on data preprocessing and modeling. First, external datasets like Pro-Football-Reference.com were not allowed, which means I couldn't include possibly important features about player and team statistics into the final model. Second, the runtime of the code could not exceed over four hours. Lastly, the evaluation process of the model is determined by this function that they call CRPS (Continuous Ranked Probability Score)

$$C = \frac{1}{199N} \sum_{m=1}^{N} \sum_{n=-99}^{99} (P(y \leq n) - H(n - Y_m))^2,$$

where N is the number of plays in the dataset, P is the predicted probability that the number of yards gained in a given play is less than n, and H is heaviside step function. The best model for this competition is chosen by whoever has the lowest score, hence my goal was to find a set of features and algorithm that will minimize this function as much as possible. For the competition, the test dataset will be running plays from the last five weeks of the 2019 season.

**Data Preprocessing**

Some of the more general data preprocessing methods that I performed for this training dataset include:

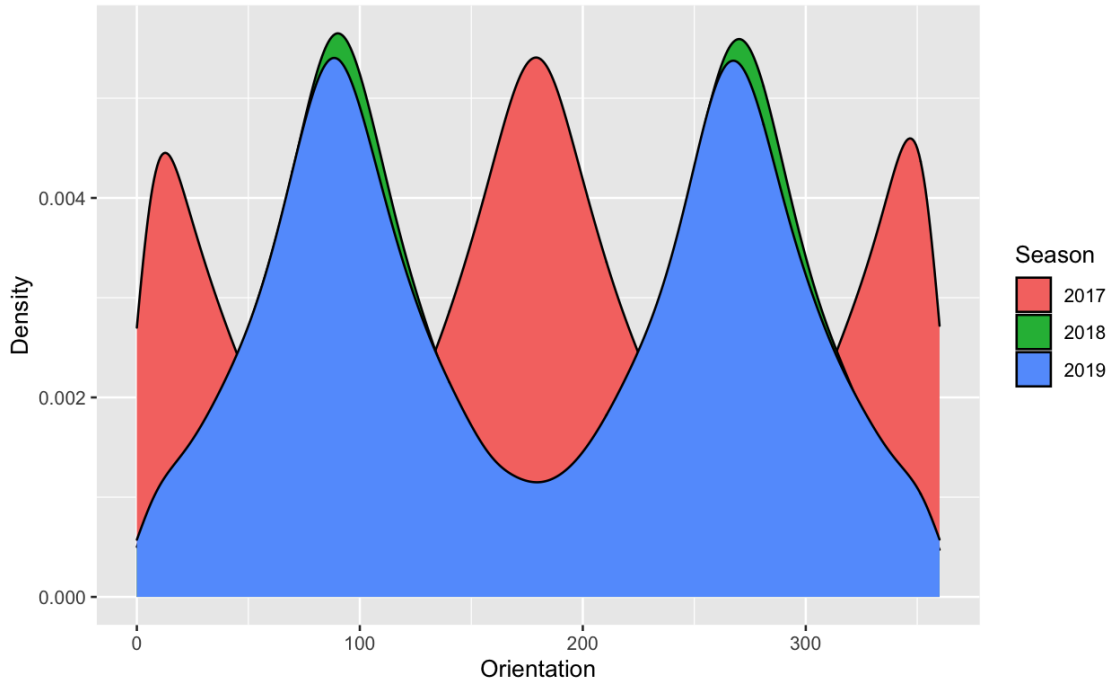
- **Handling of Missing Values:** Out of the 49 columns, 9 columns had at least one missing value as shown below in the table.

| Variable | Number of Missing Values |
|---|---|
| FieldPosition | 391 |
| OffenseFormation | 4 |
| DefendersInTheBox | 1 |
| StadiumType | 1895 |
| GameWeather | 2735 |
| Temperature | 2893 |
| Humidity | 280 |
| WindSpeed | 4176 |
| WindDirection | 4758 |

For the variable FieldPosition, those values are null if and only if the ball is in the middle of the field (Yardline = 50). To remedy this, I imputed the offensive team for those missing values. As for the remaining columns, I imputed the mean for numerical columns and mode for categorical columns to keep the process simple using SimpleImputer function from *sklearn* library [3].

- **Bucketing of Categorical Columns:** Many of the categorical columns that pertain to weather or stadiums have multiple misspellings and/or different interpretations of the same thing. For example, a game could be classified as being “Sunny” in the Weather column while another game could be classified as being “Sunny and Clear”. To fix this issue, I created general categories for these columns. For example, the Weather column was reduced from 68 unique values to having either “Clear”, “Overcast”, “Rainy”, “Snowy”, and “NA”.

- **Standardization:** An important issue that had to be addressed was that the 2017 distribution of the orientation and speed values are different when looking at 2018/2019 distributions as shown below.

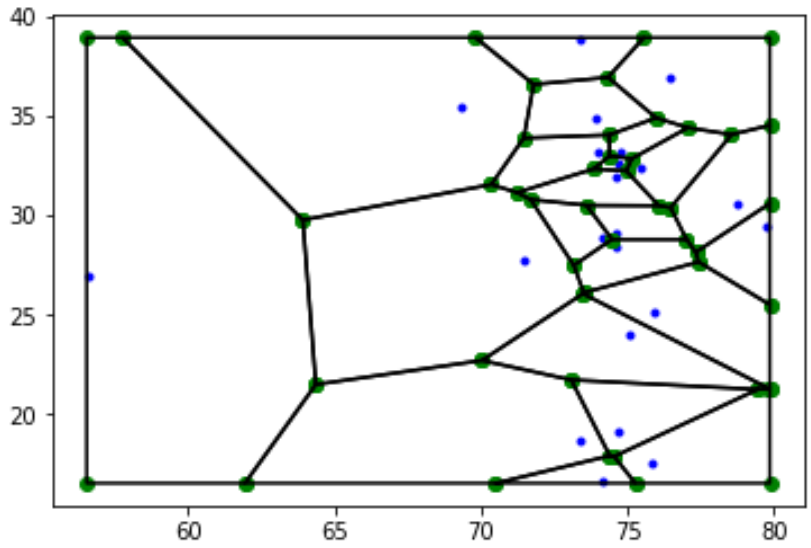



Luckily, the host of the competition provided code that standardized 2017 values to closely match 2018 and 2019, which I implemented into my own Jupyter Notebook. Also, the host of the notebook suggested that competitors standardize the position of the players and yards such that the offensive team of each play is moving in the same direction [4, 5] . This helped make things easier to analyze and model especially when doing a visualization of the play as some plays are moving left to right of the field while other plays are moving right to left.

- **Feature Engineering:** Using some of the columns available in the dataset, I engineered some potential features that could be useful in predicting the number of yards. For example, AgeOfPlayer was made using the PlayerBirthDate and DateOfGame as age

could be a useful predictor. Running backs are known for having a relatively short prime compared to other position players so older running backs might average less yards compared to newer running backs. Another example is TimeDiff which subtracts the time of snap with the time of handoff. A large time difference might indicate that it is a play action, which is known to produce high yards for running backs if done properly.

An important aspect when understanding the success of a running play is by looking at how much space the running back has. The more space the offensive line is able to provide for the running back, the higher the potential a running back will gain huge yards. If there isn't much space for the running back to operate, then the defense can easily collapse on the running back, limiting how many yards he's able to get. To apply this aspect into my model, I used voronoi diagrams for each play. A voronoi diagram partitions a plane into a set of regions such that each region is associated with a single point. The region of that particular point means that that area is the closest to that particular point compared to other points [6]. It is currently a popular method in player tracking, which helps us understand how much space a player has to operate. The figure below is an example of a voronoi diagram applied to one of the plays in the dataset.




For each play, I calculated the area of each player's region using the *scipy* library and used the quantiles of a defense's area and the area of the running back as features for the model. The final dataset includes 43 columns such as features using the voronoi diagram and the quantiles of the distance between the defender and the runner. Some of the columns were removed from the original dataset like JerseyNumber and CollegeName since those columns shouldn't have any impact on the number of yards a running back should get.

**Modeling/Results**

The two main algorithms tested for this project are Random Forests and Neural Networks.

- **Random Forest:** A classification/regression algorithm that uses multiple trained decision trees to produce an output. Whereas the main issue with decision trees is that they are very prone to overfitting, random forest fixes this by training each decision tree with a random sample (with replacement) of the data. Then, the output of each row is chosen through majority vote from these trained decision trees if classification is being done or

average value of all the trees if regression is being done [7]. This model can be found through the Python library called *sklearn*.

- **Neural Networks**: In recent years, neural networks have been increasingly gaining popularity for its breakthrough in machine learning subjects like computer vision and natural language processing. Neural nets consist of an input layer (features being fed to the neural networks), hidden layers (series of layers that perform computations on the inputs to detect patterns to produce the actual output), and an output layer (prediction using information from hidden layers). Each layer consists of nodes and these layers are interconnected with one another. This process is similar to how humans receive information with neurons [8]. This model can be found through the Python module called *keras*. For this problem, I created a Sequential model with two dense layers and a dropout layer applied after each dense layer to reduce overfitting.

Since the purpose of this competition is to find the probability that the number of yards gained in a given play is less than n, in which n ranges from -99 to 99, we will attempt to do classification on the number of yards gained for both these algorithms.

To determine which model should be used for the competition, I separated the dataset into a training set (first 26,007 plays) and a test set (last 5,000 plays). Then, I did 3-fold cross validation to train each model and determine which parameters should be used, which are shown below. The highlighted values were the ones chosen by RandomizedSearchCV from *scipy* library.

| Method | Parameters Tested | Description |
|---|---|---|
| Random Forest | • num_trees: 10, 32, 55, **77**, 100<br>• min_samples_split: **2**, 5, 10<br>• min_samples_leaf: **1**, 2, 4<br>• max_features: auto, **sqrt**<br>• max_depth: **5**, 11, 17, 23, 30<br>• bootstrap: True, **False** | • Number of trees<br>• Min. # of samples to split node<br>• Min. # of samples to be leaf node<br>• # of features to look at for each split<br>• Maximum depth of tree<br>• Whether bootstrap sample is used |
| Neural Networks | • unit (layer1): 612, **1224**, 2448<br>• unit (layer2): 306, **612**, 1224<br>• dropout_rate: 0.4, **0.5**, 0.6<br>• epochs: 5, 10, **15**<br>• learning_rate: 0.01, **0.001**, 0.001 | • # of nodes for first layer<br>• # of nodes for second layer<br>• % of inputs to be excluded<br>• # of epochs to train model<br>• Controls how fast model learns |

The table below shows the results for the CRPS score for the best parameters for each model on the training and test dataset.

| Method | Training CRPS | Test CRPS |
|---|---|---|
| Random Forest | 0.013523 | 0.013319 |
| Neural Networks | 0.012990 | 0.013309 |

Using the best parameters, neural networks had slightly better performance on the training and test datasets compared to random forest. Interestingly, the random forest model performed worse on the training set compared to the test set.

**Discussion**

Based on the results of my models, the champion model in predicting the number of yards in a given play is a neural network with two dense layers (1224 nodes for first layer and 612 nodes for second layer), a dropout rate of 0.5 applied after each dense layer, 15 epochs when

fitting the model to the dataset, and a learning rate of 0.001. With that being said, much improvements could've been made to find the best model. First, I was only able to test Random Forest and Neural Networks. However, it would be interesting to test other methods like XGBoost and see how those compare to the methods that I tested as XGBoost, for example, is known to do well in many Kaggle competitions involving classification. Second, it was difficult to test out more parameters and values due to the limitations of my computer. For example, figuring out the best parameters for neural networks took almost the whole night to run. With everyone so close to each other in the leaderboard for the competition, I think slight adjustments and further testing on the parameters could be really beneficial. Lastly, my lack of experience in neural networks/machine learning projects prior to this project and lack of time to absorb everything didn't help me either as my neural network model seems pretty simple compared to other notebooks I've looked at. With more experience with the keras library and a greater understanding in neural networks and other algorithms in the future, I think I can eventually improve on my model.

My original intention was to submit the best model to the Kaggle competition and see how well my model stacks up against others. However, I ran into multiple issues. First, I did all my coding locally into my computer, which ran fine. But when I imported my notebook to Kaggle, my code ran into many errors. My thinking is that Kaggle is running a different version of Python compared to my laptop, which are causing these errors that I wouldn't be seeing on my computer. Second, I had trouble importing the Python module *nflrush* into my Kaggle notebook, which contains the actual test dataset and Python functions to measure the code's performance. Lastly, I had little time to figure out these issues as there wasn't that much documentation about these issues and I started implementing my code to Kaggle only a few days before the deadline

of this project. In the future, I hope to figure out these issues and I will eventually be able to compare my performance against other people.

**Conclusion**

With the ever increasing use cases of machine learning and analytics applied to problems in sports, coaches and general managers are more well equipped than ever before in making sure their respective teams succeed. This study could prove beneficial to coaches and players in understanding what makes up a successful running play rather than looking at the traditional stats like yards per game. For this set of features, my neural network model was found to be slightly better than random forest in predicting the number of yards gained in a given play. With more experience in data science and a better machine, this model could be much improved upon.

**References**


1. "NFL Next Gen Stats," *NFL Football Operations*. [Online]. Available: https://operations.nfl.com/gameday/technology/nfl-next-gen-stats/.
2. "NFL Big Data Bowl," *Kaggle*. [Online]. Available: https://www.kaggle.com/c/nfl-big-data-bowl-2020.
3. F. Pedregosa, A. Gramfort, V. Michel, and B. Thirion, "Scikit-learn: Machine Learning in Python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Feb. 2011.
4. "Initial wrangling & Voronoi areas in Python," *Kaggle*, 14-Nov-2019. [Online]. Available: https://www.kaggle.com/cpmpml/initial-wrangling-voronoi-areas-in-python.
5. M. Lopez, "NFL tracking: wrangling, Voronoi, and sonars," *Kaggle*, 15-Oct-2019. [Online]. Available:

https://www.kaggle.com/statsbymichaellopez/nfl-tracking-wrangling-voronoi-and-sonars?scriptVersionId=21999507.

6. "Voronoi Diagram," *from Wolfram MathWorld*. [Online]. Available: https://mathworld.wolfram.com/VoronoiDiagram.html.
7. N. Donges, "A complete guide to the random forest algorithm," *Built In*. [Online]. Available: https://builtin.com/data-science/random-forest-algorithm.
8. A. Arnx, "First neural network for beginners explained (with code)," *Medium*, 11-Aug-2019. [Online]. Available: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf.

**Appendix**

The code for the following report can be found through this Github link: https://github.com/grantnolasco/NFLHandoff/blob/main/Stats%20295%20Project%20-%20NFL%20Handoff%20Prediction.ipynb.