NFL Play by Play Analysis: Predicting Success of an Offensive Play

Sachin Muralidharan, Caleb Sweatt

Introduction:

For our project, we aimed to build a machine learning model that could predict the success of a given NFL play using play-by-play data. To measure the success of a play, we chose a straightforward metric: yards gained. The rationale for this was simple, the more yards gained on a specific play, the more successful the play is considered to be. As massive football fans, we wanted to combine our passion for the game with the machine learning techniques we've learned this quarter. Our objective was to develop models that could accurately predict the yards gained for each play, providing valuable insights into offensive performance. By leveraging extensive NFL data and applying sophisticated machine learning methods, we aimed to create a tool that could project this key statistic and enhance our understanding of the factors contributing to successful plays.

Objectives and Design:

The primary objective of our project was to develop a predictive model capable of estimating the yards gained for an NFL play based on various factors. We hypothesized that, given accurate and comprehensive data, it is possible to create a reliable predictive model. To achieve this, we leveraged a commonly used NFL API to retrieve detailed play-by-play data spanning the past 15 seasons, from 2007 to 2022. This extensive dataset provided us with a substantial amount of information, which was crucial for training and validating our models. Our specific learning task was defined as follows: develop a neural network model to predict yards gained for each play, and compare its performance with a traditional linear regression model. The goal was to determine which modeling approach was more effective in making accurate predictions.

Python Pandas:

For our project, we used the Python Pandas library as our primary method of storing and manipulating datasets. Pandas provided an easy way to conduct data preprocessing, such as handling NaN values and transforming our data into a format suitable for linear regression and neural networks. The library's dataframe structure allowed us to easily filter and manipulate data, enabling us to quickly isolate and examine subsets of the dataset as needed. This was particularly useful for identifying and removing irrelevant plays, such as special teams plays, kneel-downs, and spikes. Pandas also facilitated various data transformations and feature engineering tasks, such as creating new features from existing data. Additionally, the library offered multiple ways to test correlations between variables and the target variable. For example, using the df.corr() method, we could efficiently compute the correlation matrix for our dataset. This allowed us to quickly see which features had the strongest linear relationships with the target variable, aiding in feature selection and model refinement. Overall, Pandas was extremely valuable in streamlining the data preprocessing and exploration process, ensuring our data was clean, well-structured, and ready for analysis.

Scikit-Learn:

In our preprocessing pipeline, we utilized the Scikit-Learn library to handle various tasks, mainly feature selections through the use of the VarianceThreshold class. The Variance threshold is an effective technique for identifying and removing features that have low variance within the dataset. This method is based on the idea that features with low variance are less likely to be useful for distinguishing between different outcomes. This tool calculates the variance of each feature in the dataset and compares it against a specified threshold. Variance is a statistical measure of the dispersion of data points in a feature and quantifies how much the values of the feature differ from the mean value. Features with low variance have values that are very similar to each other, meaning they provide little information about the differences between samples. By applying a variance threshold, we can eliminate features that are not very unique across samples. Additionally, we used the Linear regression class from the library, in order to create the regression model and compare its output with the Neural Network.

TensorFlow and Keras:

TensorFlow is an open-source machine learning framework developed by Google and provides a comprehensive ecosystem of tools, libraries, and community resources that were particularly useful for our project. TensorFlow is particularly well-suited for deep learning applications, and we used it for the neural network portion of our project. Keras, originally developed as an independent library, is now integrated into TensorFlow as its high-level API. Keras simplifies the process of building and training neural networks by abstracting many of the complexities associated with low-level TensorFlow operations, making it easier for developers to experiment with complex neural network architectures. We used dense neural networks and sequential layers for predicting yards gained in NFL plays for this project. Dense neural networks are fully connected networks, are a fundamental type of neural network where each neuron in one layer is connected to every neuron in the subsequent layer. This architecture allows the network to learn complex representations of the input data. We used the Keras Sequential API to construct our neural network model. The Sequential API enables the straightforward stacking of layers, creating a linear pipeline where data flows from the input layer to the output layer through a series of hidden layers. For our model, we decided to go with the ReLu activation function, as this is commonly used within models built for regression tasks.

Datasets and Data Preprocessing:

For this project, we utilized the nflfastpy Python library, which interfaces with the NFLfastR api to access comprehensive NFL play-by-play data dating back to 1999. We decided to use data from the 2007 to 2022 seasons, reserving the 2023 season data for testing purposes to evaluate the model's performance on new data. The data preprocessing stage was both complex and critical to the success of our project. Our initial step involved filtering out non-relevant plays, as we decided to focus exclusively on run and pass plays. We excluded special teams plays to maintain our focus on offensive performance, as these plays do not contribute directly to our model's objectives. Additionally, we removed plays where the quarterback either spiked or kneeled the ball. These plays generally occur near the end of the game and are not indicative of typical offensive performance. Plays involving penalties were also dropped, despite their potential to benefit the offense, to simplify our training process. Narrowing down the list of features from the dataset provided by the nflfastpy API was another crucial step, and was by far the most time-consuming. While we will not discuss every feature in this report, several key features were pivotal. In regards to run plays, we included directional information (left, right, middle) to capture the strategic decisions in running plays. For pass plays, we included route details and types, such as corner, post, and out, which were included to understand the patterns of passing plays. Air yards were another pivotal feature, which measured the distance the ball was thrown, providing insight into the depth of passing attempts. Lastly, game context was crucial for the model to predict plays accurately. Features like the number of seconds left in the half, the win probability of the offensive team before the play, and the yard line where the play occurred were included to capture the situational context of each play. After experimenting with the neural network using our initial feature set, we realized that the model's performance was lacking. This prompted us to engage in feature engineering, a process where we transformed existing data into more meaningful features to improve the model's performance. We created new features such as air yards against coverage types, which captured how different defensive strategies impact passing plays. To understand how the timing and route combinations influence play outcomes, we created features measuring the ratio between time to throw and the routes that occurred on the play. Play Types Given Down, Yard Line, and Game Time were incorporated to measure strategic decisions based on game context. These engineered features were designed to capture complex relationships within the data and improve predictive accuracy. Finally, a critical realization was the inclusion of contextual statistics like first_down_pass and first_down_rush, which significantly improved the model's accuracy. By engineering additional statistics from these, such as air yards on a first-down pass and rush yards on a first-down run, we concluded the massive enhancements for our feature set. Our final set of features provided essential context that was previously missing, enabling our model to make more accurate predictions. This left us with a massive 517,401 rows of data after conducting all of our preprocessing. The total number of features used in the final model was 103.

Feature	Correlation

first_down_pass_air_yards	0.682
first_down_pass	0.597
yardline_100_first_down_pass:	0.594
first_down_pass_shotgun	0.458
first_down_pass_time_to_throw	0.445
yardline_100_air_yards	0.272
air_yards	0.265
ydstogo_air_yards	0.239

Figure 1: Correlations of Features with Target Variable(Yards_Gained)

Final Results:

Model	Mean-Absolute Error	HyperParameters
Neural Network with 4 Hidden Layers, 103 Neurons	2.48	0.0001 Alpha, 50 Epochs
Neural Network with 7 Hidden Layers, 1024 Neurons	2.47	0.0001 Alpha, 50 Epochs
Linear Regression	3.17	Default

Figure 2: Table of Final Results

Interpretations and Conclusion:

From our results, we have several key takeaways. Firstly, the primary evaluation metric we chose was the mean absolute error (MAE). MAE is straightforward to interpret as it measures the average difference between the predicted values and the actual values, averaging them out across the whole dataset. A lower MAE indicates a more accurate model, whereas a high MAE suggests that the predictions deviate significantly from the actual outcomes. In our project, the MAE reflects how many yards, on average, our model's predictions were off when compared to the actual yards gained in the testing data. For testing, we used the 2023 play-by-play data, which consisted of 33,859 rows, providing a substantial amount of data to validate our model. The training dataset, spanning from 2007 to 2022, contained 517,401 rows. One of the most challenging aspects of our model was its ability to predict “big plays” (plays with 20 yards or more gained). These plays constituted only 5.4% of our training dataset, meaning the model had limited examples from which to learn. In future iterations of this project, we could employ resampling techniques to generate more randomized examples of big plays. This approach would likely enhance the model's performance and lower the MAE. Furthermore, including first-down statistics in our model, specifically, whether a team ran or passed the ball on first down, proved crucial, increasing our MAE performance by nearly 2 points. This improvement was key because the average yards gained on first-down pass plays were around 7.3 yards, while for run plays, it was approximately 4.4 yards. Our results also confirmed that air yards were one of the most significant indicators within the dataset. The model accurately predicted yards gained for passing plays where the air yards closely matched the end results. However, the model struggled with plays where players gained significant yards after the catch, resulting in higher actual yards gained. Regarding model architecture, we experimented with two different neural network structures. The first was a simpler model with four hidden layers, each containing 103 neurons, and one neuron per feature. The second model was more complex, with seven layers and a varying number of neurons: 1024 in the first layer, followed by layers with 512, 256, 200, 150, 100, 50, and 24 neurons. Despite the increased complexity, the second model only performed slightly better than the simpler one but took nearly five times longer to compile. Therefore, a simpler model was preferred for our project. Both neural network models outperformed the Linear Regression model, which had a mean absolute error of 3.17. The neural networks achieved an MAE nearly 0.7 points lower than the Linear Regression model, demonstrating their superior performance. Overall, of the 33859 rows of play by play data from 2023, the model’s predicted yards gained within 1 yard of the actual yards gained totaled to 13462 plays.

Description	Error	Yards Gained	Predicted Yards	Air Yards
4-A.O'Connell pass incomplete deep middle to 17-D.Adams [94-C.Wilkins].	0.0	0	0	41
2-D.Lock pass incomplete deep right to 11-J.Smith-Njigba (7-C.Ward).	0.0	0.0	0.0	52
(No Huddle, Shotgun) 7-G.Smith pass incomplete short middle to 84-C.Parkinson (23-D.Hill).	0.0	0.0	0.0	11.0
4-D.Prescott pass incomplete deep left	0.0	0.0	0.0	41

to 3-B.Cooks.				
22-D.Henry right tackle to CAR 1 for no gain (13-T.Hill).	0.0	0.0	0.0	0.0

Figure 3: Table of Model's most Successful Predictions

Description	Error	Yards Gained	Predicted Yards	Air Yards
(Shotgun) 9-M.Stafford pass short right to 17-P.Nacua pushed ob at NYG 2 for 80 yards	73.07	80	6.93	4.0
(Shotgun) 2-M.Rudolph pass short left to 14-G.Pickens for 86 yards, TOUCHDOWN	72.56	86	13.44	10
(Shotgun) 8-L.Jackson pass short right to 35-G.Edwards to DET 11 for 80 yards (31-K.Joseph).	69.45	80	10.55	4.0
20-Bre.Hall right guard to BUF 13 for 83 yards (47-C.Benford).	69.03	83	13.97	0
(Shotgun) 6-J.Browning pass short middle to 5-T.Higgins for 80 yards, TOUCHDOWN.	66.98	80	13.02	10.0

Figure 4: Table of Model's least Successful Predictions

Future Improvements and Use Cases:

There are several ways to improve the performance of our neural network. Implementing resampling techniques for specific types of plays can enhance the model's ability to predict those plays accurately. Additionally, incorporating more detailed defensive characteristics and traits could provide a more comprehensive understanding of play outcomes. Including more information about offensive schemes and systems could highlight the nuances between different offensive styles in the NFL, and could further improve the model's predictive power. Despite these areas for improvement, our model’s performance, with an MAE of approximately 2.48, is promising and lays a strong foundation for future enhancements. In terms of practical applications, NFL teams could use this neural network to model offensive game plans for specific games. By analyzing which types of plays generally succeed and which do not, teams can optimize their play-calling strategies. Furthermore, sports bettors could leverage this model to analyze the tendencies of specific teams, helping them make more informed betting decisions. For instance, they could use the model to evaluate whether a team’s offensive game plan is likely to be effective against a particular opponent’s defense. Overall, while there is room for improvement, our model provides valuable insights and a robust foundation for predicting successful NFL plays. This has the potential to be a powerful tool for both NFL teams and armchair sports analysts.

References:

[1] “What It Takes To Win.” John Hamman (Stanford CS229, Fall 2011)

[2] Beal, Ryan & Norman, Timothy & Ramchurn, Sarvapali. (2020). A Critical Comparison of Machine Learning Classifiers to Predict Match Outcomes in the NFL. International Journal of Computer Science in Sport. 19. 10.2478/ijcss-2020-0009.