Predicting the Outcome of NFL Games using Machine Learning

John O'Connell

Abstract

This project explores the feasibility of predicting the outcome of NFL games using several machine learning algorithms. The algorithms that will be explored are standard linear regression, ridge regression with various basis function expansions, Huber loss regression, and deep neural networks. The deep neural networks explored are feed forward neural networks with different combinations of layers, nodes, and activation functions. The data used for this project was scraped from ProFootballReference.com and consists of team statistics and game outcomes for the past 13 seasons in the NFL. The models explored produced an average accuracy in predicting the winner of 70.59%, and an average against the spread winning percentage of 62.88%.

Introduction

The National Football League (NFL) is a multibillion-dollar industry, and one of the biggest professional sports leagues in the world. Gambling on the NFL dates to the 1930s, when both legal and illegal gambling was sweeping across the nation (Rolfe, 2023). Even back then, bookmakers were looking for ways to predict the point spread (a number that serves as a handicap between two opponents) for games. More recently in 2018, the Supreme Court overturned the federal ban on sports betting outside of Nevada, opening the door for legal sports betting nationwide (Waters, 2023). Since that time, an estimated $100 billion has been wagered at licensed sportsbooks during each NFL season ("How Much Money", 2023). Given the high stakes involved in NFL gambling, sportsbooks need a way to accurately predict the points spread of games. While traditional statistical methods have been used in the past to predict game outcomes, the recent advancements in machine learning have opened new opportunities for developing more accurate and sophisticated prediction models. This project aims to leverage machine learning and the extensive amount of readily available data to successfully predict the outcome of NFL games. The successful outcome of this project will be beneficial to both the sportsbook setting the gambling lines for NFL games as well as those gamblers attempting to outsmart the sportsbooks and profit from betting on said games.

Background

In order to appreciate the metrics used to judge the success of this project, it is necessary to understand the basics of gambling on NFL games. More specifically, it is crucial to understand the idea of the point spread. The point spread is a popular technique used in sports betting to balance out the perceived skill levels of the teams and create more even betting odds. It is a number that represents the amount of points

that one team is favored by over the other. For example, if the New England Patriots are favored by 7 points in a game against the Buffalo Bills, it can be said that the point spread is New England -7. Conversely, the point spread can also be given in terms of the underdog. In the previous example, the point spread can also be said to be Buffalo +7. When placing a bet on an NFL game, a gambler can choose to bet on the team that is favored to win (the "favorite") or the team that is expected to lose (the "underdog"). If the bettor chooses the favorite, they must win by a margin greater than the point spread to cover the spread and win the bet. If the bettor chooses the underdog, they can win the bet even if the team loses, as long as the final score difference is less than the point spread.

Three different metrics were used to judge the success of the models in this project. The first was the winning percentage of the model's predictions against the point spread. If the model predicted the home team would cover the spread, and they did in fact cover, it was considered a win. Conversely, if the model predicted the away team would cover the spread, and they did in fact cover, it was also considered a win. Another important thing to note is that gambling lines are typically not set at even money, meaning that whatever a person wagers is not necessarily the same amount they will win. The most common gambling line on a point spread in the NFL is -110, implying that a person would have to risk $110 to win $100. This is significant because it is not enough for the model to have a winning percentage above 50%. In order to be profitable, it can be shown that the winning percentage must be at least 52.4% (Kontos, 2018). The second metric used to judge the success of the models was the accuracy of predictions. This simplifies the winning percentage metric and only looks at whether the correct winner was predicted, regardless of any point spread. The final metric used was the Mean Squared Error (MSE) of the model's predictions. MSE is a standard performance metric used to evaluate the accuracy of a regression model.

Summary of Data

The data for this project was collected by writing Python scripts that utilized the BeautifulSoup library to scrape data from ProFootballReference.com. ProFootballReference is a popular website for NFL statistics and history. All data collected, as well as the final processed data used in the models can be found at the following link: Data for Project. Season long statistics for all 32 NFL teams for the past 13 years were collected, which equates to 416 tuples of data. Each tuple consists of 68 features for a total of 28,288 data points. Histograms showing the distribution of all 68 statistics collected can be seen in the Appendix of this report. Additionally, the outcome and final point spread of every regular season NFL game for the past 13 seasons were collected, a total of 3,360 games.

There was additional data preprocessing required after scraping in order to obtain the desired input vectors as well as output scalar for this project. In order to simplify the data, the statistics used for each game as well as the outcome of each game was put in terms of the home team. The input for each game is an average of the home team's offensive stats with the away team's defensive stats, as well as an average of the home team's defensive stats with the away team's offensive stats. The outcome of a game is

the final score in terms of the home team's point differential, and the given point spread was altered when necessary to always reflect the home teams point spread. Representing all data and results in terms of the home team made it easier to evaluate the performance of each model. Finally, the input data for all models was scaled using a MinMax Scaler. A MinMax Scaler subtracts the minimum value of the feature and then divides by the range. The range is the difference between the original maximum and original minimum.

Model Evaluation

The first model explored was a simple linear regression model using the SciKitLearn package in Python. For this model 10 seasons worth of data, or 2,560 games, were used for training and 3 seasons worth, or 799 games, were used for testing. This works out to approximately a 76/24 train-test split. Below is the cost function associated with simple linear regression:

$$J(\theta) = \sum_{i=1}^{N} (y_i - \theta^T x_i)^2 \; = \; \|Y - X\theta\|_2^2$$

Where Y is the given results vector, X is a matrix of input vectors and θ is the feature vector. This model performed surprisingly well given that it is the most straightforward of all models tested. The results of the model can be seen in Table 1.

Table 1: Results of Simple Linear Regression Model

	Total Num Games	Winning Bets Count	"No Bet" Count	Winning Percentage	Winner Correctly Predicted Count	Accuracy in Classification	MSE	RMSE
Training Data	2560	1542	104	62.79%	1833	71.60%	143.16	11.96
Testing Data	799	480	35	62.83%	547	68.46%	140.58	11.86

One important thing to note in the results is the "No Bet" column. When the model predicted a home team point differential that was the same as the official point spread of the game, it was counted as a no bet and not factored into the winning percentage. This is true for all models explored during this project. Overall, a 62.83% against the spread winning percentage and 68.46% accuracy on the testing data indicates a very successful and profitable model. Although the MSE may seem somewhat high at 140.58, the number of factors in play when determining the exact score of an NFL game is exceedingly high. To average inside of two touchdowns (14 points) of the final score for every game across three years is impressive.

Due to the aforementioned difficulty of predicting results, it was hypothesized that there would be many outliers in the data. Therefore, the next model explored was Huber regression, which utilizes the Huber loss cost function. The Huber loss cost function is a robust regression algorithm that assigns less weight to observations identified as outliers. The loss function associated with Huber loss regression can be seen below:

$$L_\delta(y_i - \theta^T x_i) = \begin{cases} \frac{1}{2}(y_i - \theta^T x_i)^2, & |y_i - \theta^T x_i| \le \delta \\ \delta|y_i - \theta^T x_i| - \frac{\delta^2}{2}, & otherwise \end{cases}$$

Once again, this model was implemented using the SciKitLearn package, and the same 76/24 train-test split of the data. In order to run Huber regression, a value must be assigned to the hyperparameter delta (δ). The value of delta controls the balance between robustness to outliers and smoothness of the loss function. A larger value of delta results in a loss function that is more robust to outliers, but less smooth and less sensitive to changes in the parameters. To determine the optimal delta for the data, the model was trained with delta values ranging from 1 to 4. The results of training were plotted and the model with the best delta value was chosen. Plots of the training results can be seen in Figure 1.

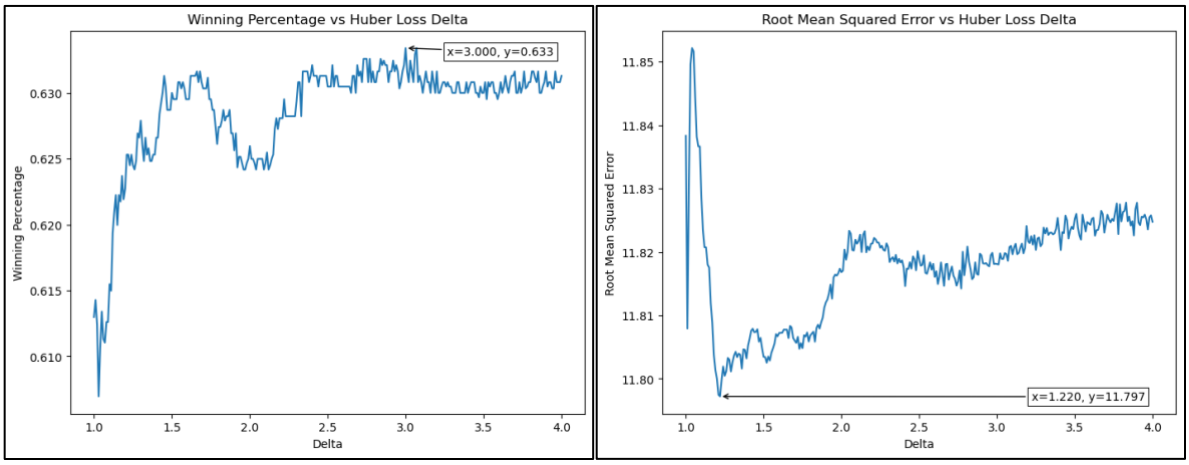


Figure 1: Plots of Model Results vs Huber Loss Delta

A different optimal delta was found for different evaluation metrics. For the sake of this project, the winning percentage against the spread is more important than the MSE or RMSE, and therefore a delta value of 3 was chosen for the Huber regression model. The results of the Huber regression model with a delta value of 3 can be seen in Table 2.

Table 2: Results of Huber Regression Model

	Total Num Games	Winning Bets Count	"No Bet" Count	Winning Percentage	Winner Correctly Predicted Count	Accuracy in Classification	MSE	RMSE
Training Data	2560	1566	105	63.79%	1834	71.64%	141.71	11.90
Testing Data	799	489	27	63.34%	552	69.09%	139.67	11.82

Somewhat surprisingly, the robustness to outliers did not have a large effect on the results of the model. The Huber regression model performed only slightly better than the standard linear regression model, with a 63.34% against the spread winning percentage and 69.09% accuracy on the testing data.

The next model explored was a linear regression model utilizing basis function expansion. Basis function expansion is a technique to capture nonlinear relationships in the data. In basis function expansion the original features are transformed into a set of new features, which are created by applying a set of basis functions to the original features. The cost function associated with basis function expansion can be seen below:

$$J(\theta) = \sum_{i=1}^{N} (y_i - \theta^T \varphi(x_i))^2 \; = \; \|Y - \emptyset\theta\|_2^2$$

The only difference between this cost function and that of standard linear regression is the phi term in place of x. This phi term represents the basis function expansion of the features in x. Once again, this model was implemented using the SciKitLearn package, and the same 76/24 train-test split of the data. When running the model with different basis functions, it was experiencing quite a bit of overfitting. It would perform exceptionally well on the training data but poorly on the testing data. Because of this, the decision was made to switch to a ridge regression model to control the overfitting. Ridge regression modifies the cost function by adding a penalty term that regularizes the results, reducing the variance of the estimates and helping to avoid overfitting. The new cost function associated with the ridge regression model and basis function expansion can be seen below:

$$J(\theta) = \sum_{i=1}^{N} (y_i - \theta^T \varphi(x_i))^2 \; + \; \lambda \sum_{j=1}^{d'} \theta_j^2 = \; \|Y - \emptyset\theta\|_2^2 + \; \lambda\|\theta\|_2^2$$

Where lambda is a hyperparameter that controls the strength of the penalty term. The basis functions explored include polynomials up to degree two, sine and cosine functions, and a combination of the two. The results of the ridge regression model can be seen in Table 3.

Table 3: Results of Ridge Regression Model with Basis Function Expansion

		Total Num Games	Winning Bets Count	"No Bet" Count	Winning Percentage	Winner Correctly Predicted Count	Accuracy in Classification	MSE	RMSE
Polynomials Up To Degree 2	Training Data	2560	1655	108	67.50%	1896	74.06%	121.11	11.01
	Testing Data	799	470	14	59.87%	524	65.58%	168.76	12.99
Linear, Sine, Cosine	Training Data	2560	1569	100	63.78%	1839	71.84%	140.24	11.84
	Testing Data	799	493	24	63.61%	551	68.96%	141.53	11.90
Polynomials + Sine and Cosine	Training Data	2560	1701	95	69.01%	1909	74.57%	112.92	10.63
	Testing Data	799	463	17	59.21%	521	65.21%	177.21	13.31

Some overfitting is still present in the basis functions that include second degree polynomials as they perform significantly better on the training data vs testing data. The sine and cosine functions produce the best results, and slightly outperform the previous models seen with a 63.61% against the spread winning percentage and 68.96% accuracy on the testing data.

The final model explored was a feed forward neural network using the PyTorch package in Python. For this model 8 seasons worth of data (2,408 games) were used for training, 2 seasons of data (512 games) were used for validation, and 3 seasons of data (799 games) were used for testing. There are many parameters that can be changed when implementing a feed forward neural network. The neural network used in this project utilized a Mean Squared Error loss function, a Stochastic Gradient Descent optimizer, and 1000 epochs during training. The parameters changed during model exploration were the number of layers (1 vs 2 layers), the nodes per layer (68 vs 136 nodes), and the activation function used between layers (Tanh vs ReLU vs Sigmoid). The results of all variations of the neural network implemented can be seen in Table 4.

Table 4: Results of Feed Forward Neural Network Models

		Train Winning %	Train Accuracy %	Train MSE	Test Winning %	Test Accuracy %	Test MSE
Tanh Activation Function	1 Layer, 68 Nodes Per Layer	62.03	71.78	148.2613	62.73	69.46	137.8644
	1 Layer, 136 Nodes Per Layer	62.91	71.58	146.843	62.53	69.84	137.2317
	2 Layers, 68 Nodes Per Layer	62.73	71.63	145.9355	62.2	69.59	137.3056
	2 Layers, 136 Nodes Per Layer	62.64	71.48	148.815	63.46	69.96	137.858
ReLU Activation Function	1 Layer, 68 Nodes Per Layer	62.77	71.53	147.3079	62.47	69.71	137.3135
	1 Layer, 136 Nodes Per Layer	63.17	71.53	146.0824	61.99	70.09	137.374
	2 Layers, 68 Nodes Per Layer	63.42	71.48	144.9771	61.47	69.96	137.442
	2 Layers, 136 Nodes Per Layer	63.49	71.68	145.8529	62.14	69.21	136.9565
Sigmoid Activation Function	1 Layer, 68 Nodes Per Layer	62.75	71.34	147.0568	62.92	69.71	137.4281
	1 Layer, 136 Nodes Per Layer	62.58	71.58	148.6212	64.11	69.59	137.6663
	2 Layers, 68 Nodes Per Layer	62.65	71.39	147.9969	63.02	70.59	137.8434
	2 Layers, 136 Nodes Per Layer	62.89	71.53	147.6591	63.1	69.96	137.4813

The parameters explored had very little effect on the performance of the neural network. All neural network models performed similarly to the linear regression variations previously seen. The one area in which they differ slightly is the MSE. The MSE of the neural networks was higher than previously seen when evaluated on the training data, and lower than previously seen when evaluated on the testing data. Ultimately however, the difference is not large enough to prompt further exploration. Given that no one neural network performed better than the rest, it is best to keep the model as simple as possible as doing so will ultimately reduce computation time and resources. The training and validation losses per epoch for one of the simple models, a network with 1 layer, 68 nodes per layer and the Tanh activation function can be seen in Figure 2.

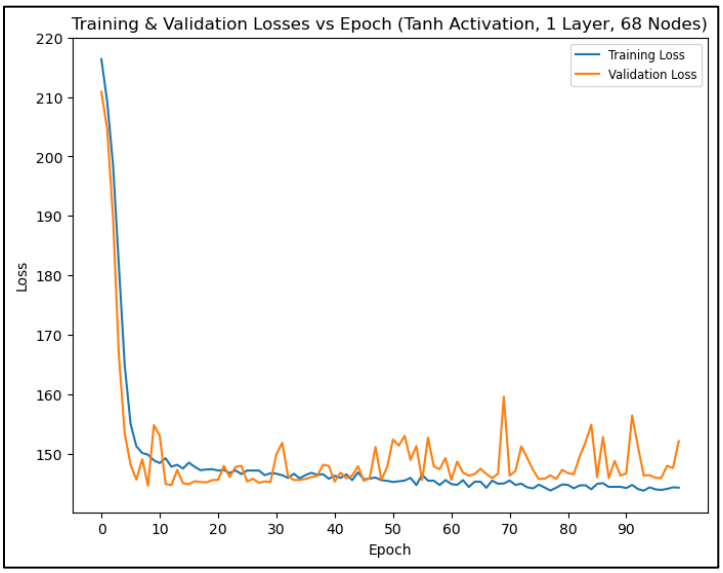


Figure 2: Training & Validation Losses Plotted vs Epoch

Overall, all models explored for this project performed similarly when the correct parameters were selected. The average against the spread winning percentage was 62.88% – far above the 52.4% needed to be profitable. Additionally, the average accuracy in predicting a winner was 70.59% and the Mean Squared Error was in the range of 137-143.

Related Work

Attempting to predict the outcome of NFL games for gambling purposes is nothing new. However, doing so in an academic setting is not as popular, and only a few examples of such work was found during research. Of the five examples found, all of them attempt to predict the outright winner of the game, but only two attempt to predict the against the spread winner. The machine learning models explored in this project outperformed all other attempts to solve this problem. The most accurate model attempting to predict an outright winner of the game was made by Beal, Norman and Ramchurn in their paper “A Critical Comparison of Machine Learning Classifiers to Predict Match Outcomes in the NFL”. They were able to achieve an accuracy of 67.53% using a Naïve Bayes classifier. The machine learning models in this project produced an average accuracy in predicting a winner of 70.59%, which outperforms that of the Naïve Bayes classifier. The most accurate model attempting to predict against the spread winners was found by Jim Warner in his paper “Predicting Margin of Victory in NFL Games: Machine Learning vs. the Las Vegas Line”. Warner was able to achieve an against the spread winning percentage of 50.90% using a Gaussian process model. Once again, the machine learning models in this project produced an average against the spread winning percentage of 62.88%, which far outperforms the Gaussian process model. The full list of related work can be found in the references section of this report.

Conclusion

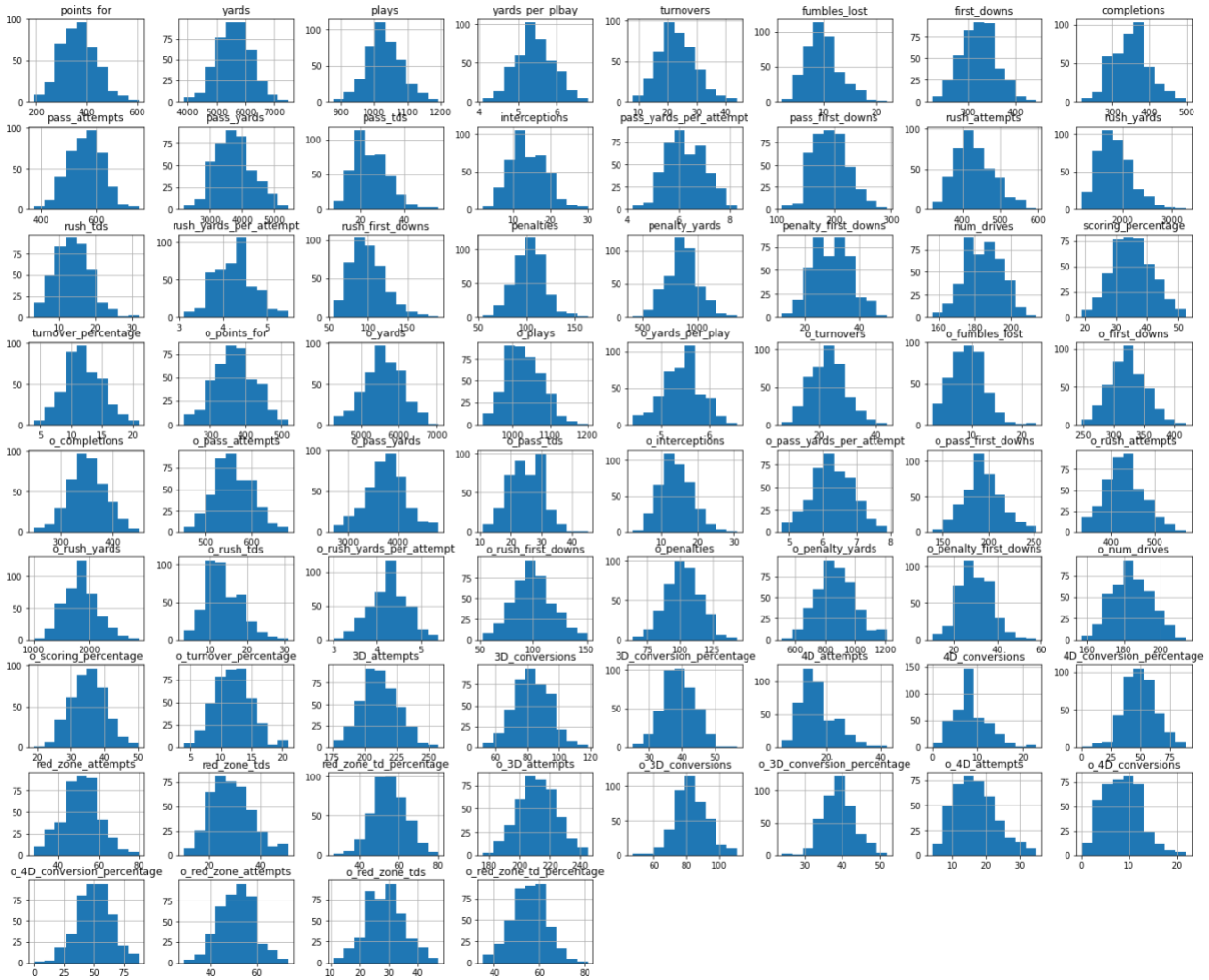
This project demonstrated the ability for machine learning algorithms to successfully predict the outcome of NFL games, both outright and against the spread. The models explored during this project produced an average against the spread winning percentage of 62.88%, and an average accuracy in predicting an outright winner of 70.59%. To put the against the spread winning percentage in perspective, let's say an individual placed 100 bets against the spread of $110 each. As mentioned previously, the most common gambling line on a point spread is -110, meaning that a winning bet of $110 would return $100. Rounding up to a 63% win percentage for simplicity, this means that the individual would win 63 of their 100 bets for a total winnings of $6,300. Subtracting $4,070 for the 37 lost bets of $110 each, the final winnings would be $2,230. This is an extremely high rate of return for 100 bets and demonstrates the successfulness of the machine learning models.

In the future, it would be interesting to further explore the features used in the models. Additional features such as more specific or informative statistics can be added to see their effect on the predictions. Additionally, running lasso regression may be useful to see which of the current features have little or no effect on the predictions and can be removed. Furthermore, from a gambling perspective the results of the models can be studied to see if certain prediction ranges lead to losing bets. These results can then be removed to produce a higher winning percentage.

Overall, this project was very successful in demonstrating the usefulness of machine learning algorithms in NFL game predictions. Real world in season testing is needed to confirm that the models will produce similar results in practice. If they hold up in practice, these models will prove to be an extremely useful tool when gambling on NFL games.

Appendix

Figure 3. Histograms of all 68 statistics collected per team per year



References

How much money do Americans bet on sports. LegalSportsBetting.com. (2023, February 10). Retrieved March 3, 2023, from https://www.legalsportsbetting.com/how-much-money-do-americans-bet-on-sports/

Kontos, K. (2018, August 7). *Break-even win % for sports betting*. BettingPros. Retrieved April 20, 2023, from https://www.bettingpros.com/articles/break-even-win-for-sports-betting/

Pro Football Stats, History, Scores, Standings, Playoffs, Schedule & Records. Pro. (n.d.). Retrieved March 3, 2023, from https://www.pro-football-reference.com/

Rolfe, B. (2023, February 1). *A brief history of betting on the NFL*. Pro Football Network. Retrieved March 3, 2023, from https://www.profootballnetwork.com/a-brief-history-of-betting-on-the-nfl/

Waters, M., & Candee, A. (2023, March 3). *Sports betting revenue tracker - US betting revenue & handle by State*. Legal Sports Report. Retrieved March 3, 2023, from https://www.legalsportsreport.com/sports-betting/revenue/

Related Work References

Beal, R., Norman, T., & Ramchurn, S. (2020). *A Critical Comparison of Machine Learning Classifiers to Predict Match Outcomes in the NFL.* University of Southampton.

Bosch, P. & Bhulai, S. (2018, February). *Predicting the winner of NFL-games using Machine and Deep Learning.* Vrije Universiteit Amsterdam. (Unpublished).

Finkel, R. (2020, August). *Predicting National Football League Games Against the Point Spread.* Utica College. (Unpublished).

Hamadani, B. *Predicting the outcome of NFL games using machine learning*. Stanford University. (Unpublished).

Warner, J. (2010, December 17). *Predicting Margin of Victory in NFL Games: Machine Learning vs. the Las Vegas Line*. (Unpublished).