Beat The Bookmaker:
Identifying Whether the Use of Machine Learning Models Can Accurately Predict NFL Scores and Make Better Bets

Submitted for the Degree of Master of Science in

Data Science and Analytics

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

August 29th, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 11,729

Student Name: Shreyan Nageswaran

Acknowledgements

With this project marking the conclusion of my academic career at Royal Holloway, University of London, there are several people I would like to thank for their assistance throughout this journey. First being my project supervisor for his guidance throughout the duration of this project. Next being my family for their continuous support and confidence in me, and for being an example of what it means to accomplish anything with hard work and dedication.

Abstract

The use of Machine Learning models to predict the scores of Association Football matches have proven to be both effective and well-known in the realm of predictive analysis for sports. However, the translation of these various models between sports, such as American Football, is an open frontier to experiment. As a result, this project aims to test the accuracy of Machine Learning models, such as a Poisson Distribution as well as a Decision Tree Regression Model, in predicting outcomes for both Regular Season and Playoff matches in the National Football League. This project also aims to further test the capabilities of Machine Learning models by creating model-derived betting probabilities for each match and comparing them against odds from leading bookmakers.

Contents

1 Introduction

1.1 Motivation

The predictive power of machine learning models has captivated the attention of Association Football fans, leading to the development of a multitude of experiments on the sport. The results of these experiments further appeal to followers of the sport who seek to “take a scientific, mathematical approach to betting” [1]. As a result, Association Football has become synonymous with the use of a Poisson distribution model to both predict match outcomes and place more accurate bets.

This phenomenon caused me to question the validity of these experiments, as it seems unlikely that an algorithm can be used to predict an outcome controlled by humans. After all, athletes are the ones who dictate the result of their respective sporting match. Furthermore, regarding the betting approaches, utilizing models to enhance betting returns could greatly impact an individual's financial situation. Therefore, to gauge the accuracy of these models, an attempt was made to replicate these experiments. However, seeing that there are numerous existing implementations of these experiments for Association Football, it seemed enticing to try and recreate these experiments on a different sport such as American Football.

The motivation to work on this project also stems from the desire to improve on skills that will be useful in industry. Working on this project will allow me to accumulate more experience in the realm of programming, machine learning, and research which will greatly help me in future career opportunities.

1.2 Project Objective and Procedure

The objective of this project is to see if the machine learning algorithms, used in Association Football, can be used to both accurately predict American Football scores and help bettors place more accurate bets. Furthermore, this project also aims to see if other machine learning models, outside of the ones used for Association Football, can be used to generate more accurate predictions. This experiment will be conducted by first designing a Poisson Distribution model utilizing explicit calculations for each parameter. After an evaluation of the results, the next step will be a comparison with a derivation of the popularly used methodology for predicting Association Football fixtures. Then, the experiment will

consider additional parameters used in Association Football procedures and look to utilize them in the context of American Football. Once the Poisson Distribution model experiment has concluded, the next step is to evaluate the performance of an entirely different machine learning model, a Decision Tree Regressor. From there, the results presented from the variations of the Poisson model as well as the Decision Tree Regressor will serve as an indicator for how well machine learning models can predict American Football scores. However, the experiment continues with an analysis of betting odds. The odds from the dataset are converted into predicted probabilities, which can then be used to calculate how accurate odds are in predicting the outcome of a game. This will allow for a direct comparison between how well the model did in its predictions compared to the odds as well as either assert or refute the idea that machine learning models can be used to make more accurate bets.

1.3 Betting Industry

As betting odds are a focal point of this project, it is important to understand the industry and its significance. Sports betting has a long history dating back to the Industrial Revolution [2]. Since then, it has become increasingly popular and has had a large impact on several economies. For example, sports betting, among others in the American Gaming Association, "contributed $328.6 billion to the U.S. economy in 2023" [3]. An economic contribution of that magnitude indicates that many people partake in sports betting and place high value bets. In fact, in that same year, "Americans legally wagered $120 billion just on sports betting" [3].

Despite the sports betting industry having beneficial effects on the economy and being a source of entertainment, many people lose significant amounts of money due to inaccurate bets being placed. Since 2018, Americans have surrendered approximately $240 billion in sports betting [4]. This shows the significant toll that sports betting can have on an individual's finances, and, as a result, it is unwise to promote the usage of a machine learning algorithm to make bets if it is not proven to be accurate.

This project does not encourage gambling, however, it simply aims to see if machine learning models can also be used to make more accurate bets, as models are being used in that capacity for Association Football.

2 Background Research

2.1 Poisson Model

The Poisson Distribution model is the primary machine learning model that will be implemented and researched during this project. A Poisson Distribution model is described as a "discrete probability distribution" [5], which means that it can be used to identify the "probabilities of occurrence of different possible outcomes in an experiment" [5]. In comparison to other machine learning algorithms, a Poisson Distribution model is the ideal choice to generate predictions for a given event. One of the main advantages of utilizing a Poisson Distribution is its simplicity. Based on its equation, the Poisson Distribution requires few parameters.

$$P(k \text{ events in interval}) = e^{(-\lambda)} * ((\lambda^k)/k!)$$

Figure 1: The Poisson Distribution Equation

Equation sourced from [5]

To fully comprehend the mechanics of the model, it is essential to understand the significance of each variable in the equation. The equation takes in two variables, k and ƛ. P(k events in interval) is "the probability of observing k events in a given interval" [5]. Based on this interpretation, it can be inferred that k can be changed based on the desired number of occurrences. The ƛ denotes the "event rate" [5] which is the "expected number of possible occurrences" [5]. These variables, coupled with "e", the Euler's number [5], are used to calculate the P(k events in interval).

Both variables, k and ƛ, can be easily applied in a variety of scenarios and that is what makes the Poisson Distribution model appealing in the realm of Association Football. In the context of Association Football, the k and ƛ variables correspond to values that influence the predicted score of a match. Data Scientist, David Sheehan, explicitly outlines how the ƛ parameter can be mapped to the expected number of goals scored by a team [6]. In this interpretation of the ƛ parameter, Sheehan also illustrates how to calculate it in the context of Association Football.

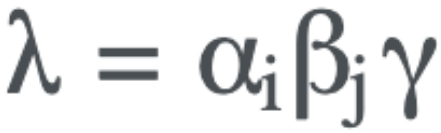


Figure 2: The Lambda Equation

Image sourced from [6]

In the equation above, Sheehan utilizes two additional parameters to calculate the ƛ. The first parameter, a, represents the attack strength of the first team [6], the second parameter, b, represents the defense strength of the second team [6], and the third parameter, y, represents the home field advantage [6]. These three parameters combined make up the ƛ. However, the most important takeaway from this is how the lambda can be adjusted depending on the teams in each matchup. These additional parameters ensure that the ƛ is tailored to each team's performance in a given match.

In this context, k can be associated with a scoring event [7]. To visualize what these parameters would look like in an Association Football setting, a variation of the scenario depicted by Elena Petrova, in her article on CareerFoundry [5], would be most helpful. In her article, Petrova provides an example in which she articulates the values of the ƛ and k, where she states that, given the average number of internet failures, what is the probability of observing three failures [5]? In her example, ƛ represents the average number of internet failures and k represents three failures [5]. Considering what is now known regarding the interpretation of these parameters for Association Football, a similar situation can be created. For example, given that Arsenal averages three goals per game, what is the probability of them scoring four goals in their match against Chelsea? In this context, the ƛ represents the average goals scored by Arsenal per game and the k represents four goals. This provides an overview as to how these parameters are used in the context of Association Football.

2.2 Dixon Coles

The Dixon Coles Parameter is a unique addition to the Poisson Distribution model to further enhance its accuracy. In an effort to improve on the Poisson Model, Mark Dixon and Stuart Coles identified two cases where the model was lacking: the accuracy when predicting low scoring matches and the equal weightage of both recent and older matches [6].

To rectify the first issue, Dixon and Coles sought to implement a parameter that "applies a correction" [6] on the model, which is referred to as the "rho" parameter [6]. Below is the set of equations in which the rho parameter is calculated.

$$\tau_{\lambda,\mu}(x,y) = \begin{cases} 1 - \lambda\mu\varrho & \text{if } x = y = 0 \\ 1 - \lambda\varrho & \text{if } x = 0, y = 1 \\ 1 + \mu\varrho & \text{if } x = 0, y = 1 \\ 1 - \varrho & \text{if } x = y = 1 \\ 1 & \text{otherwise} \end{cases}$$

Figure 3: Set of equations used to calculate rho

Image sourced from [6]

There are numerous ways in which rho can be calculated depending on the low scoring match that occurs. From this set of equations, it is determined that only scores ranging from 0 to 1, however, are considered for the usage of this parameter. This implies that scores above this range are not considered to be low scoring matches.

Regarding the second issue, one of the ways to fix the equal weightage of both recent and older matches is to "only consider matches within some predefined period" [6]. However, as outlined in later sections, this concept is insignificant in the context of American Football.

2.3 Decision Tree Regressor

Decision Tree Regressors are one of the most fundamental machine learning algorithms. A Decision Tree Regressor is an algorithm that "constructs a tree-like model to predict continuous numerical values" [8], which makes it applicable in the context of predicting American Football scores. When analysing the workflow of a Decision Tree, we can see the intricate steps taken in order to generate predictions.

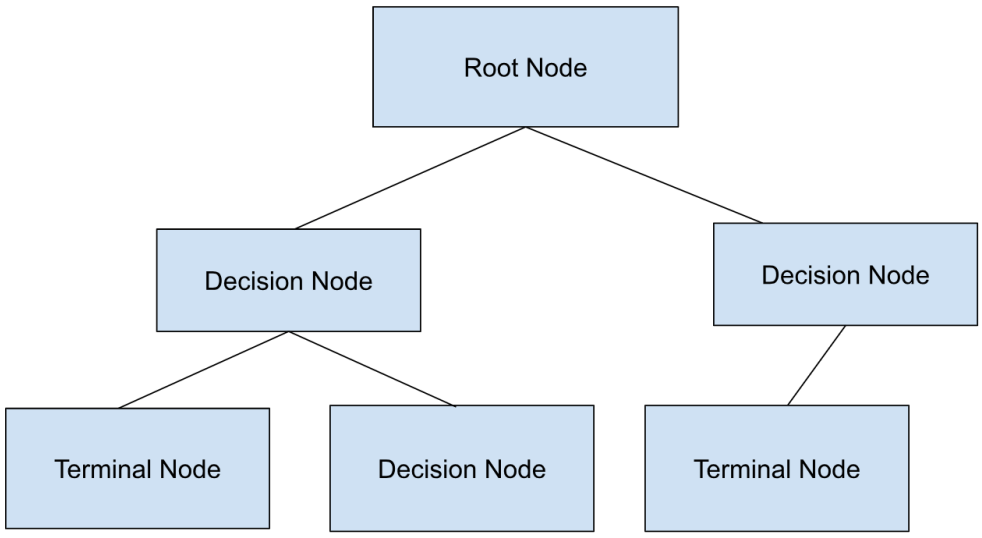


Figure 4: Diagram of Decision Tree Process

Image inspired from [9]

The diagram above illustrates the Decision Tree process from the start to the completion of the algorithm. The Decision Tree starts at the root note, which "represents the whole data points" [9]. From there, the tree splits into various decision nodes. Each decision node expresses a decision that needs to be made [10]. Branching from the decision nodes, are terminal nodes, also referred to as leaf nodes [9], which represent the outcome of those decisions [10].

There are several important factors to consider in regard to Decision Trees. One of the most important concepts in the Decision Tree Process is the splitting of nodes. A Decision Tree diverges from the root node, and further splitting occurs on other nodes, based on a "variable at each step that best splits the set of items" [9]. Additionally, another significant concept to note from this procedure is the fact that this process is not linear, meaning that decision nodes can also be further split into other decision nodes before reaching a terminal node.

In the context of this project, a Decision Tree is being used as an additional algorithm to verify the results generated from the Poisson Distribution model. Though it is unlikely that both models will generate identical results, utilizing a Decision Tree Regressor will provide insight as to whether other machine learning models, outside of the Poisson Distribution model, can be used to predict sports scores. When searching for an additional model to cross reference with, there were certainly other

models that were appealing to use in this project such as a Random Forest. A Random Forest is similar to a Decision Tree, in that it is also a "tree-based machine learning algorithm" [11], however, it utilizes the power and capabilities of more than one Decision Tree in order to generate predictions [11].

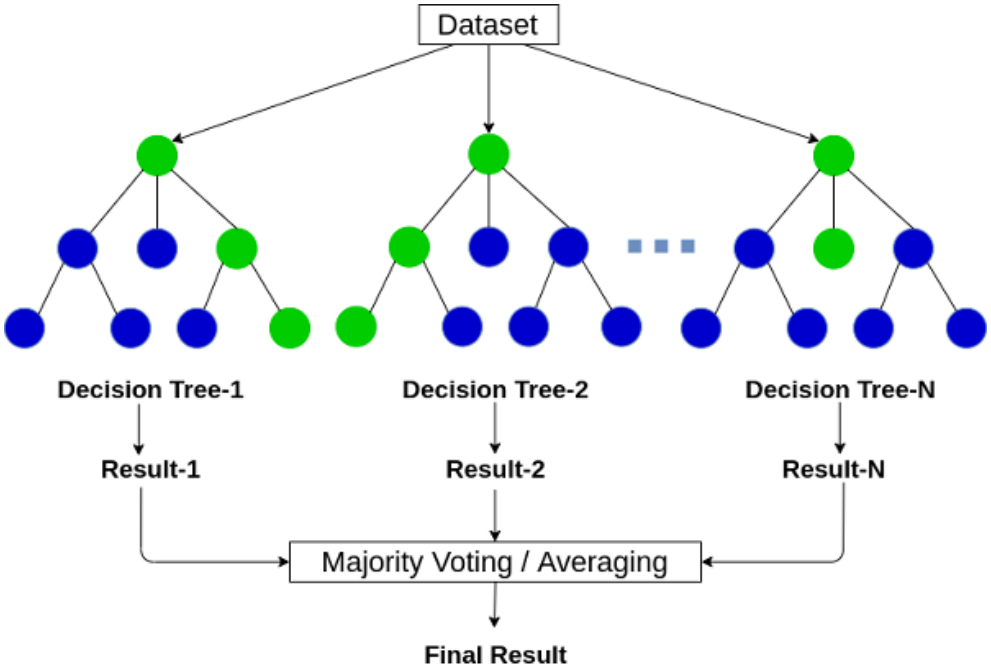


Figure 5: Diagram of Random Forest Procedure

Image sourced from [11]

A Random Forest model is able to generate its result through an accumulation of outputs from a Decision Tree. This calls into question whether a Random Forest would be a better suited algorithm for this project compared to using a single Decision Tree. However, the main advantage in using a single Decision Tree is that it is faster as opposed to a Random Forest [11]. This is crucial as, when utilizing large datasets consisting of historical data, a Decision Tree will run quicker whereas a Random Forest algorithm could take significantly more time.

2.4 American Football and Association Football Differences

Despite their identical names, American Football and Association Football are very different sports and, to successfully create a derivation of the Association Football inspired Poisson Model for American Football, it is

imperative to discern the differences between them. The most notable difference between the sports is in how they are scored. In Association Football, the procedure is straightforward. The scoring metric in Association Football is a goal, which occurs "when the entire ball crosses the goal line between the goalposts and under the crossbar" [12]. The team that scores the most number of goals, in an Association Football match, is declared the winner [12].

The scoring procedure in American Football, however, is more complex. In contrast to Association Football, there are numerous ways in which a team can score points. The most significant scoring metric in American Football is a touchdown. In American Football, a touchdown occurs when a player in possession of the ball crosses the opponents' goal line [13]. When this happens, the team that scored the touchdown is rewarded with 6 points [13]. In addition to this, after scoring a touchdown, a team can elect to try for either a single point or an additional 2 points [13]. This means that a team can score from 6 to 8 points as a result of scoring a touchdown. Furthermore, if a team is unable to score a touchdown but has strong field position, they can opt to kick a field goal which would result in a score of 3 points [13]. Though these scoring metrics are predominantly scored by a team's offense, there are ways in which a defense can score points for their team as well. In the event that a defense pushes the other team's offense into their own goal, this would result in a score of 2 points and is called a safety [13]. At the end of the match, like Association Football, the team that accumulated the higher score is declared the winner.

Given that goals in Association Football only account for a single point, this results in significantly lower scoring matches in comparison to American Football. After an analysis of approximately 294,970 matches, a study found that the most common scoreline in Association Football was 1-1, which implies that only two goals were scored in the entirety of the match [14]. In contrast, for American Football, scores are significantly greater due to the variety of ways in which a team can score points. In fact, a similar study for American Football found that the most common scoreline was 20-17 [15], implying that multiple scoring plays occurred for both teams throughout the match. This difference plays an integral role in the construction of the Poisson Model, as well as the varying parameters to consider, as the model must be able to accurately predict notably larger scores than those that it would traditionally see in the case of Association Football. As a result, certain parameters that were used to predict Association Football scores were unable to be fully utilized in this experiment such as the Dixon Coles term. Additionally, the stark differences in scores directly affected how the predicted scores were calculated compared to Association Football experiments, which will be discussed in the upcoming sections.

3 Data Collection

3.1 Project Data

For this project, the collection of data was centered around the type of information that was needed to be included. With the scope of this project, it became apparent that the most essential component of data would be historical results of American Football scores, more specifically, National Football League scores from previous years' regular season and playoffs. Acquiring this data would allow for comparisons of model-generated predictions to real-world predictions and aid in the computations for the various model parameters. Furthermore, to compare model-generated win probabilities against betting odds, it was imperative to also include a form of historical betting odds for each game into the datasets.

3.2 Data Scraping and Dataset Construction

The first step to acquiring the data was to identify a source that contained all the components that were necessary for this project. After numerous searches, I came across the page, sportsoddshistory.com [16], which consisted of both past games in American Football as well as their corresponding betting odds, in the form of spreads. With this information, I then began manually scraping the data from the webpage into my datasets. This allowed me to structure my dataset in a coherent manner. The objective was to construct the datasets by having each team's schedule (with their opponents and the score for each match) as well as the betting spread for them winning the match. As a result, the datasets consisted of 7 columns. The first two columns, 'League' and 'Date', were used to simply identify when each match occurred. Though these columns were not used in the experiments, it was necessary to include them to ensure that each team, and their score, were corresponded to the correct game when inputting the data. The next four columns, 'Team1', 'Team2', 'T1 Score', and 'T2 Score', represented the teams in each match as well as the scores for 'Team1' and 'Team2'. The final column in the dataset is 'T1 Spread', which is the spread odd for the team, in the 'Team1' column. It is important to note that, in these datasets, the same teams and scores do repeat. However, the order changes, meaning that the team in 'Team1' gets shifted to 'Team2', the team in 'Team2' get shifted to 'Team1', the score in 'T1 Score' moves to 'T2 Score', and the score from 'T2 Score' moves to 'T1 Score'. This is to ensure that the values in the 'T1 Spread' column reflect the odds for each team in a match. Though this approach may be less efficient, it thoroughly guarantees that each team and game is accounted for in the dataset.

4 Poisson Experiment

4.1 Overview

The Poisson Experiment was conducted for two full seasons in the National Football League, the 2018 and 2019 regular seasons and playoffs, and consisted of using two variations of the model. In the first implementation, the construction of the model parameters was done through mathematical computations, whereas, in the second implementation, the model parameters were built through optimization techniques. Given that the optimization method is widely used in models for Association Football, it is worth seeing which approach will net more accurate results. Overall, this experiment not only tests the validity of the Poisson Distribution model for American Football, but also evaluates both approaches and determines which is more accurate.

4.2 Variation #1: Mathematically Derived Parameters

The first step in implementing this variation was to construct a training and test set. Given the nature of an American Football league season, in which there consists of a regular season and a playoffs, it seemed logical to assign the regular season scores as the training set and the playoff scores as the test set. Once this was completed, the next task was to define the parameters that will be used in the model. As mentioned in Section 2, in the context of Association Football, the λ parameter represents the expected number of goals that a team would score in a match and is comprised of the attack strength, defense strength, and home field advantage [6]. Though, in the context of American Football, not all of these fields within the λ parameter are useful. Since there are direct correlations for attack strength and defense strength in American Football, these fields were used.

Even though it has been deduced that the offensive strength and defensive strength parameters will be used in this project, that is still not enough to accurately estimate λ. Prior implementations of the Poisson Model for Association Football calculated the expected goals scored (λ) by taking e^x in which x is a sum of the attack and defense strengths [17]. That approach does not work within the framework of American Football due to the fact that, as outlined in Section 2, the scoring for both these sports is different. As a result, that approach will result in an expected score that is not feasible for an American Football match. However, another way to calculate the expected score, as outlined in the article "How to calculate Poisson distribution for football betting" on help.smartmarkets.com, is

through multiplying both strengths together along with the average points scored by a team [18]. The article outlines that, the calculation of the expected score for the home team, for example, is "Home team attack strength * away team defense strength * average number of home goals" [18]. Similarly, an extraction of this equation can be applied to American Football, resulting in the equation:
Expected Points for Team 1 = Team 1 Offensive Strength * Team 2 Defensive Strength * Team 1 Average Points Scored.

With the framework to calculate the λ parameter, the next step was to compute the values of the offensive, defensive and average point scored metrics. In similar fashion to the λ equation, a derivation of equations used to calculate the attack and defensive strengths in Association Football can be used to build the offensive and defensive strength parameters in this experiment. In Association Football, the attack strength for a home team can be calculated as "home team's average goals per home game / average home league goals per game" [18]. This can be translated into the equation to construct the offensive strength parameter in this model as: Team 1 Offensive Strength = Team 1 Average Points Scored per game / League Average Points Scored per game. For defense strength, the aforementioned article defined defensive strength for a home team as "Home team's average goals conceded per home game / average home league goals per game" [18]. In the context of this project, this can be conveyed as: Team 1 Defensive Strength = Team 1 Average Points Allowed per game / League Average Points Allowed. Below is the pseudocode of the implementation for the offensive and defensive strength parameters in the project.

```
Function buildInitialOffParams(data)
        Create empty dictionary for offensive parameters
        Create empty list representing allTeams
        For each row in data
                If team1 not in allTeams
                        Add to allTeams
                If team2 not in allTeams
                        Add to allTeams
        For each team in allTeams
                Set offensive parameter to 0
Return offensive parameter dictionary
```

Figure 6: Pseudocode for the *buildInitialOffParams* method

```
Function buildInitialDefParams(data)
        Create empty dictionary for defensive parameters
        Create empty list representing allTeams
        For each row in data
                If team1 not in allTeams
                        Add to allTeams
                If team2 not in allTeams
                        Add to allTeams
        For each team in allTeams list
                Set defensive parameter to 0
Return defensive parameter dictionary
```

Figure 7: Pseudocode for the *buildInitialDefParams* method

It was imperative to first initialize the parameter for each team by setting it to a temporary placeholder, which is 0. This ensured that a dictionary was constructed for both offensive and defensive parameters with pre-existing values inside. Though these functions may seem insignificant, considering that the placeholder values will have no effect on the outcome of the parameter construction, the primary motivation for creating these functions was to build a dictionary, in which keys are team names, so that it would be simpler to map the correct parameter to each team.

```
Function calculateOffensiveMetrics(data):
        Create empty dictionary for average scores per team
        Create empty dictionary for number of games per team
        Create empty list representing allTeams
        For each row in data
                If team1 not in allTeams
                        Add to allTeams
                If team2 not in allTeams
                        Add to allTeams
        For each team in allTeams
                Assign 0 to value average scores, number of games
        For each row in data
                And for each team in allTeams
                        If team equals team in row's 'Team1' column
                                Add the team's score to value of the team in average scores
                                Increment number of games by 1
        For each key, value pair in the average scores dictionary
                Divide value by number of games for team
Return the average scores dictionary
```

Figure 8: Pseudocode for the *calculateOffensiveMetrics* method

```
Function calculateDefensiveMetrics(data):
    Create empty dictionary for average scores per team
    Create empty dictionary for number of games per team
    Create empty list representing allTeams
    For each row in data
        If team1 not in allTeams
            Add to allTeams
        If team2 not in allTeams
            Add to allTeams
    For each team in allTeams list
        Assign 0 to value average scores, number of games
    For each row in data
        And for each team in allTeams list
            If team equals team in row's 'Team1' column
                Add the other team's score to value of the team in average scores
                Increment number of games by 1
    For each key, value pair in the average scores dictionary
        Divide value by number of games for team
Return the average scores dictionary
```

Figure 9: Pseudocode for the *calculateDefensiveMetrics* method

As defined earlier, in order to construct the parameters, the average points scored and allowed for each team must be acquired, which is the objective of these functions. With these four methods implemented, the process of updating the offensive and defensive parameters can begin.

```
Function updateParameters(data, offensive parameters, defensive parameters):

    Create empty list representing allTeams
    For each row in data
        If team1 not in allTeams
            Add to allTeams
        If team2 not in allTeams
            Add to allTeams

    Create empty dictionaries for total games, total points

    For each team in allTeams
        Assign 0 to value total games, total points

    Call calculateOffensiveMetrics, set returned values in off_stats
    Call calculateDefensiveMetrics, set returned values in def_stats

    Set league_avg variable to 0
    For each row in data
        And for each team in allTeams
            If team is equal to row's 'Team1' column
                Add team score to value of team in total points
                Increment total games by 1

    Set league_points variable to 0
    For each key, value pair in total points
        Add value to league_points

    Set all_games variable to 0
    For each key, value pair in total games
        Add value to all_games

    league_avg = league_points/all_games

    For each key, value pair in offensive parameters
        Divide team's offensive stat by league_avg

    For each key, value pair in defensive parameters
        Divide team's defensive stat by league_avg

    Create empty dictionary called fitted_params
    For each key, value pair in offensive parameters
        Add to fitted params a key, 'offense_' + key, and value off_params[key]

    For each key, value pair in defensive parameters
        Add to fitted params a key, 'offense_' + key, and value off_params[key]

    Return fitted_params
```

Figure 10: Pseudocode for the *updateParameters* method

To compute the parameters, the first step was to retrieve the offensive and defensive statistics from the methods described earlier. Then, to calculate the league average points scored and allowed per game, I iterated through the dataset and retrieved the values of points scored by each team. Given that the total points scored in the league is equivalent to the total points allowed in the league, it was only necessary to retrieve the values in one of the score columns. The summation of all values in that column represents both the total number of points scored and the total number of points allowed in the league. Furthermore, to find the average, a similar calculation was needed to retrieve the total number of games played by all teams in the league. After deriving both these values, the league average points scored and allowed metric was calculated by dividing the total number of points in the league by the total number of games.

With all the metrics, average points scored for each team, average points allowed for each team, and league average points scored and allowed, there was sufficient information to calculate our offensive and defensive strength parameters using the equations defined earlier.

Team	**Offensive Parameter**	**Defensive Parameter**
Buffalo Bills	0.7202141900937081	1.001338688085676
Baltimore Ravens	1.0414993306559572	0.7684069611780455
Cincinnati Bengals	0.9852744310575636	1.2182061579651942
Detroit Lions	0.8674698795180723	0.963855421686747
Minnesota Vikings	0.963855421686747	0.9129852744310576
New Orleans Saints	1.3493975903614457	0.9451137884872824
New York Giants	0.9879518072289156	1.1030789825970548

Figure 11: Example of Calculated Parameters for the 2018 season

Team	**Offensive Parameter**	**Defensive Parameter**
Buffalo Bills	0.8602739726027397	0.7095890410958904
Baltimore Ravens	1.4547945205479451	0.7726027397260274

Cincinnati Bengals	0.7643835616438356	1.1506849315068493
Detroit Lions	0.9342465753424658	1.158904109589041
Minnesota Vikings	1.115068493150685	0.8301369863013699
New Orleans Saints	1.2547945205479452	0.9342465753424658
New York Giants	0.9342465753424658	1.2356164383561643

Figure 12: Example of Calculated Parameters for the 2019 season

The parameters above are examples of the calculations for both the 2018 and 2019 datasets. A larger offensive strength indicates that the team had a better than average offense. For example, in 2019, the New Orleans Saints offensive strength was approximately 1.25. This implies that the Saints had an above average offense whereas a team such as the Detroit Lions, who finished with an offensive strength of approximately 0.93, had a below average offense. In the case of the defense strengths, the opposite applies. The lower the defense strength is, the better the actual defense is. For example, in 2018, the Baltimore Ravens had a defense strength of approximately 0.76, which implies that they were an elite defense that season. On the contrary, the Atlanta Falcons, in that same season, had a defense rating of approximately 1.13, which implies that they were worse than league average.

4.3 Game Predictions

With the parameters established, we can now begin predicting the scores for each game. To utilize these parameters, I created a method used to predict the outcome of each game.

```
Function gamePrediction(team1, team2, params, avg_off, maxTDs, data):

    Set team1_offense variable to params['offense_' + team1]
    Set team1_defense variable to params['defense_' + team1]
    Set team2_offense variable to params['offense_' + team2]
    Set team2_defense variable to params['defense_' + team2]
    Set team1_avg_off variable to avg_off[team1]
    Set team2_avg_off variable to avg_off[team2]

    Set game_points variable to maxTDs * 6

    Calculate team1_exp_points as team1_offense * team2_defense * team1_avg_off
    Calculate team1_exp_points as team2_offense * team1_defense * team2_avg_off

    Set team1_prob list values to poisson probability mass function of
    sorted(range(game_points)), team1_exp_points

    Set team2_prob list values to poisson probability mass function of
    sorted(range(game_points)), team2_exp_points

    #Plotting Code for Team 1#

    Set value of predicted_score1 variable to 0
    Set value of prbs1 variable to 0
    For each index in range of team1_prob
        If value at team1_prob[index] > prbs1
            Assign index to predicted_score1
            Assign team1_prob[i] to prbs1

    #Plotting Code for Team 2#

    Set value of predicted_score2 variable to 0
    Set value of prbs2 variable to 0
    For each index in range of team2_prob
        If value at team2_prob[index] > prbs2
            Assign index to predicted_score2
            Assign team2_prob[i] to prbs2

   Set win1_overall_prob variable to 0
   Set win2_overall_prob variable to 0

   Create empty list called win_prbs1
   Create empty list called win_prbs2

   For each i in range(len(team1_prob))
       And for each j in range(len(team2_prob))
           If i > j
               Add team1_prob[i] * team2_prob[j] to win_prbs1
           Otherwise
               Add team2_prob[j] * team1_prob[i] to win_prbs2

   Set win1_overall_prob to sum of all values in win_prbs1
   Set win2_overall_prob to sum of all values in win_prbs2

   #Visual Print Statements#

Return predicted_score1 and predicted_score2
```

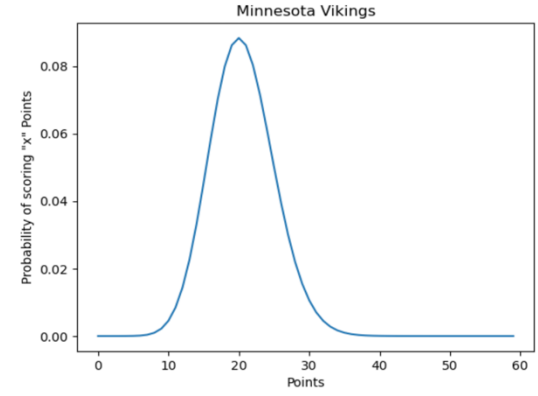
Figure 13: Pseudocode for the *gamePrediction* method

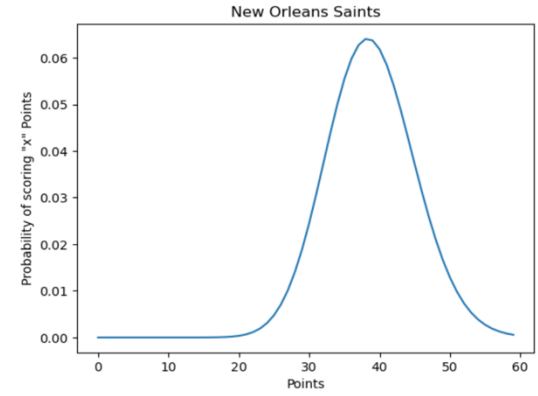
The lines of code up until 'Plotting Code' are inspired by [17]

Similarly to how predicting a match is done for Association Football implementations, the first step was to retrieve all the parameters defined beforehand and use them to calculate the expected points for both teams in the match [17]. From there, we can utilize the Poisson Distribution model in order to retrieve the probabilities of scoring a certain number of points [17]. In these initial lines of code, one of the main differences is the usage of the *game_points* variable. In the implementations for Association Football, the writers utilize a *max_goals* variable and pass that into the Poisson Distribution model to get the probabilities of scoring each number of goals until it reaches the *max_goals* [17]. However, since American Football scoring is significantly different, we can modify this variable to be

maxTDs (which represents maximum touchdowns) and, before passing it into the model, multiply it by six to represent the maximum number of points scored in the game.

Given that the Poisson Distribution model returns a list of probabilities with the same size as the *game_points* variable, the predicted score is the score that returns the highest probability, calculated by iterating through the list and identifying the index in which the probability is the greatest. In this case, the index refers to the predicted score. The next step was to calculate the overall predicted win probability for each team. In the case of Team 1, for example, if Team 1 was projected to score x points and Team 2 was projected y points, and x > y, then the probabilities of both those events occurring would be multiplied together. This results in the probability of Team 1 winning the match with that exact score. The overall win probability for each team is calculated by summing the probabilities, in which they are predicted to score more points than the opponent, together.






```
Match Prediction: Minnesota Vikings: 20  vs  New Orleans Saints: 38

New Orleans Saints expected win probability = 98.94%
Minnesota Vikings expected win probability = 0.67%
```

Figure 14: Visual representation of a game prediction

Now that the model is able to generate a prediction for each sporting match, the next step was to see how accurate its predictions were in comparison to the real results from that year. This could be gauged through the calculation of an accuracy rate variable. The accuracy rate variable was computed by counting the occurrences in which the predicted winner was the actual winner and then dividing by the total number of predictions.

```
Function evaluateModel(dataset, params)

    Set accuracy rate variable to 0.0
    Set correct predictions variable to 0
    Set total predictions variable to 0

    Create empty list representing completed matches

    For each row in dataset
        Assign team1 and team2 to teams in columns 'Team1' and 'Team2'
        Call calculateOffensiveMetrics, set returned values in offense
        Call calculateDefensiveMetrics, set returned values in defense

        If team in row's 'Team2' and 'Team1' columns not in completed matches
            Call gamePrediction, set returned values in predicted_score
            Add match to completed matches

            Get actual team scores from row's 'T1 Score', 'T2 Score' columns

            differential1 = team1 predicted score - team1 actual score
            differential2 = team2 predicted score - team2 actual score

            #Visual Displays of Predicted Winner and Differentials#

        If model predicts correct winner
            Increment correct predictions, total predictions by 1

        Otherwise
            Increment total predictions by 1

        Calculate accuracy rate as correct predictions/total predictions * 100

        #String Display of Accuracy Rate#

Return accuracy rate string
```

Figure 15: Pseudocode for *evaluateModel* method

4.3.1 2018 and 2019 Season Experiments

For the 2018 Regular Season, which is denoted as the training set, the model was 67.97% accurate in identifying the winner in each match. When using the same training parameters on the test set, which is the 2018 Playoffs, the model returned an accuracy rate of 63.64%.

For the 2019 Regular Season, the model generated a 67.77% accuracy rate. However, when using the same training parameters, the model

surprisingly predicted 90.91% of games correctly in the corresponding test set, which is the 2019 Playoffs.

The biggest surprise when analysing this data is how much higher the accuracy rate is for the 2019 test set as opposed to the other training sets or even the 2018 test set. Though this may seem inconsistent, there are a few reasons as to why this could have occurred. The main reason is due to the test set size. In the dataset containing the scores for the regular season, there are approximately 512 rows of data. However, the dataset containing the playoff scores only contains about 22 rows of data. This means that any incorrect predictions on the test set contribute more to a decrease in accuracy rate in comparison to incorrect predictions on the training set. This is likely the cause for the decrease in accuracy rate from the training set in 2018 to its corresponding test set. Conversely, a smaller test set gives the model less opportunities to make incorrect predictions, which explains why the test set accuracy rate in 2019 was substantially greater than the rates for the training set.

Overall, given that the accuracy rates returned by the model in this implementation are consistently greater than 60%, this proves that, with mathematically derived parameters, a Poisson Distribution Model can be used to accurately predict American Football games in most cases.

4.4 Variation #2: Optimized Parameters

The second version of this experiment will be performed by utilizing optimization techniques in order to calculate the offensive and defensive strength parameters for the model. Since, in this approach the parameters are not being calculated based on the prior equations, as defined earlier, functions such as *buildInitialOffParams*, *buildInitialDefParams*, *calculateDefensiveMetrics*, and *updateParameters* are not needed in this implementation. However, *calculateOffensiveMetrics* still needs to be included due to the fact that the expected number of points scored will be determined based on a product of the offensive and defensive strengths generated along with the average points scored by a team. With that, along with our training and test sets being defined in the same manner as they were in the previous variation, the optimization process can commence.

4.4.1 Code References

The functions used for optimizing the parameters: *log_likelihood*, *fit_poisson*, and *fit*, are sourced from the article, "Predicting Football Results With The Poisson Distribution", on pena.lt/y [17], with minor adjustments in order to utilize those functions for American Football. Since I wanted to draw a comparison between the mathematically derived parameters approach and the optimized parameter approach, I looked to replicate the optimization

procedure in order to ensure that this variation of the model would be represented accurately in the comparison.

4.4.2 Implementation

As outlined in the article, in experiments like this, the typical approach to optimization is to utilize Maximum Likelihood Estimation [17], which means that "the parameters are chosen to maximize the likelihood that the assumed model results in the observed data" [19]. However, since likelihood calculations involve extensive computations with small numbers [17], that may not be the most optimal method. Instead, the better choice would be to "minimize the negative of the log-likelihood" [17].

```
Function log_likelihood(team1_score, team2_score, team1_offense, team1_defense,
team2_offense_, team2_defense, team1_avg_off, team2_avg_off)

    Calculate team1_expected as team1_offense * team2_defense * team1_avg_off
    Calculate team2_expected as team2_offense * team1_defense * team2_avg_off

    Set value of team1_llk to poisson probability mass function of (team1_score,
    team1_expected)

    Set value of team2_llk to poisson probability mass function of (team2_score,
    team2_expected)

    If team1_llk > 0 and team2_llk > 0
        Set value of variable log_llk to sum of log of team1_llk and log of team2_llk
    Otherwise
        Return 1

Return -log_llk
```

Figure 16: Pseudocode *log-likelihood* method

Inspired by [17]

Similar to how it is used in the Association Football implementation article [17], this log-likelihood method calculates the expected scores as well as the log-likelihoods for each team and returns the negative sum of both team's log-likelihoods.

The next stage of this procedure is to create our *fit_poisson* function to generate the values for the optimized parameter.

```
Function fit_poisson(data)

        Create empty list called teams
        For each row in data
                If team1 not in teams
                        Add to teams

        For each row in data
                If team2 not in teams
                        Add to teams

        Set value of number of teams variable to length of teams

        Create empty list called params1
        Create empty list called params2

        For each value in teams
                Add random value to params1

        For each value in teams
                Add random value to params2

        Set value of params variable to params1 + params2

        Call calculateOffensiveMetrics, set returned values in offenses

        Function fit(params, data, teams, offenses)

                Create offensive_params dictionary pairing teams and parameter value
                Create defensive_params dictionary pairing teams and parameter value

                Create empty list called loglikes
                For each row in data
                        Call log_likelihood, add returned value to loglikes

        Return sum of values in loglikes

    Call minimize function, set returned values in result

    Create model parameters dictionary pairing offense and defense team labels with
    parameter result

Return model parameters
```

Figure 17: Pseudocode for *fit_poisson* and *fit* functions

Inspired by [17]

Again, similar to how it is implemented in the Association Football implementation [17], the *fit_poisson* function initially starts by generating a set of random parameters for both offensive and defensive strength. Then, the *fit* function, which is written inside of the *fit_poisson* function, is used to iterate through our dataset and add up all the log-likelihoods for each game [17]. The final step is to then utilize the minimize function, on the *fit* function with the random parameters defined, to generate the optimized parameters for the model's use, which will then be stored in a dictionary [17].

The parameters generated from these functions are significantly different in comparison to the parameters derived in the first variation of the model. For example, the offensive strength for the Buffalo Bills, in 2018, was calculated to be approximately 9.13. An offensive strength of this magnitude would indicate that the Bills had an exceptional offense that season. This value differs significantly from the offensive parameter computed for them in the previous implementation, which was approximately 0.72, that indicated that their offense was worse than league average. In addition to this, other teams have significantly different parameters than in the previous experiment. These changes in parameters should definitely impact the overall performance of the model. Once the parameters are created, we can utilize the same *gamePrediction* and *evaluateModel* functions to make our predictions and calculate the accuracy rate.

4.4.3 2018 and 2019 Season Experiments

For the 2018 training set, which is the Regular Season, the model accurately predicted the winner in 50.39% of games. When analysing the 2018 playoffs, the model netted an accuracy rate of 59.09%.

For the 2019 training set, the model returned an accuracy rate of 72.46%, and, for the 2019 test set, the model generated an accuracy rate of 81.82%.

Though these results may seem rational, if not more or less impressive, in comparison to the accuracy rates generated by the first variation of the model, there is a clear indication as to why the first variation is more consistent as opposed to this one. When looking at the line of code with the minimize function (in the second implementation), the parameters that get passed into the minimize function are random. Though this may seem insignificant, it proves that the model will consistently generate different parameters even with the use of the same training set. As a result, there may be significant fluctuations in predicted scores and accuracy rates the more times that this algorithm is run. For example, if the code is run again, the parameters generated will be different which, in turn, implies that the accuracy rate will change. To verify this, we can run the algorithm again and see the variations in results between both cases.

When running the code for the algorithm again, there are immediate changes in both the parameters and the overall accuracy rates. Starting with parameters, there are major differences compared to the ones generated in the initial execution of the code. For example, for the 2018 datasets, the Buffalo Bills returned an offensive parameter of 0.61. This is significantly lower compared to the offensive parameter generated in the initial execution of the code, which was approximately 9.13. Furthermore, when examining the accuracy rates, it is apparent how the overall rates

differ in comparison to the ones generated from the initial execution. The model returned accuracy rates of 41.02% and 68.18% on the 2018 training and test set. For 2019, the model returned accuracy rates of 58.2% and a shockingly low 27.27% on the training and test set.

From the results returned in the second execution of the code, it can be concluded that this version of the model will not yield a consistent accuracy rate over several executions of the code. Given this revelation, it seems illogical to denote this approach as the 'better approach' since there is no concrete measurement of accuracy that will be consistent throughout numerous iterations of the code.

4.5 Summary

Overall, both variations of the Poisson Distribution Model do seem to be able to accurately predict American Football scores to an extent. However, implementing the model with mathematically derived parameters will result in more consistent results.

4.6 Dixon Coles Inspired Parameter

The Dixon Coles Model has been commonly linked to the Poisson Model when predicting Association Football scores. However, for American Football, the Dixon Coles model has several shortcomings that prevent it from being fully used in this experiment. As mentioned in the earlier sections, the Dixon Coles model is used to better approximate predictions for low scoring matches in which either zero or one goals are scored [6]. In American Football, the scoring system prevents games from having scores this low. Even if we consider low scoring games, in American Football, as games where both teams scored a single touchdown or less, this approach still would not be effective as, through an examination of the 2018 dataset, there was only one game in which both teams scored six or less points. In addition to this, the second goal of the Dixon Coles model, in which games are weighted based on their recency [6], also does not apply in the domain of American Football since there are significantly fewer games in a season compared to Association Football. In the 2018 National Football League season, for example, each team only played 16 games [20]. In an Association Football league, such as the Premier League, each team plays 38 games [21]. As a result, for American Football, every game should be weighted equally in predicting a team's performance given the smaller sample size of games.

With the lack of compatibility between American Football and the Dixon Coles model, it seems as though the Dixon Coles model is not practical for this project. However, it is possible to implement the principles of the

Dixon Coles model in another capacity. Considering the primary goal of the model, which is to improve accuracy for low scoring games, a derivation of the Dixon Coles model can be implemented by identifying low scoring games in the context of American Football. We can further improve this derivation by also identifying abnormally high scoring games so that the new parameter can be used to better approximate both low and high scoring games.

Given the guidelines for the new parameter, the next step would be incorporating it in the implementation of the Poisson Distribution model. Since the Poisson Distribution model with the mathematically computed parameters generates consistent accuracy rates, we can integrate the new parameter there to clearly see how it impacts the accuracy of the model. The first step was to identify any games that were anomalies, meaning games where teams either scored an abnormally high or low amount of points. This was determined by observing matches in which both teams either scored below 7 points, above 49 points, or did not score at all.

```
Function identify_anomalies(data)

    Create empty list representing anomaly games
    For each row in data
        If team1 score < 7 and team2 score < 7
            Add teams to anomaly games
        If team1 score > 49 and team2 score > 49
            Add teams to anomaly games
        If team1 score == 0 and team2 score == 0
            Add teams to anomaly games

    Create empty list called non duplicates
    For each pair in anomaly games
        If reverse pair not in non duplicates
            Add pair to non duplicates

Return non duplicates
```

Figure 18: Pseudocode for the *identifyAnomalies* method

The next step was to initialize these 'anomaly' parameters. Resembling how the offensive and defensive parameters were initialized, a dictionary was constructed containing each team and their parameter value, which in this case was set to 1.0. The reason for setting it to 1.0 as opposed to a random value is because not all teams should be impacted by this parameter. For example, if a team has never had a poor or exceptional scoring effort, their expected score should not be impacted by this new parameter. Initializing each team's anomaly parameter to 1.0 ensures that teams that have had no anomalies are not susceptible to changes in their expected score.

```
Function buildInitialAnomalyParameters(data)
        Create empty dictionary representing anomaly parameters
        Create empty list representing allTeams

        For each row in data
                If team1 not in allTeams
                        Add to allTeams
                If team2 not in allTeams
                        Add to allTeams

        For each team in allTeams list
                Set anomaly parameter value to 1.0

Return anomaly parameters
```

Figure 19: Pseudocode for the *buildInitialAnomalyParameters* method

After the initialization, the next step was to update these new parameters. Unlike the offensive and defensive strength parameters, there is not a concrete formula to use to construct these. Hence, logical reasoning was used to build these parameters. If a team consistently had low scoring anomalies, their expected score should be lightly reduced. Conversely, if a team consistently had high scoring anomalies, their expected score should slightly increase to take these into account. Based on this premise, the anomaly parameter was first calculated by acquiring the amount of anomalies as a fraction of the total number of games played. If the anomaly was for a high scoring game, this would be positive, but for a low scoring game this would be negative. For example, the Denver Broncos, in the 2019 dataset, had three games in which they scored zero or less than seven points. As a result, their anomaly parameter was approximately -0.1875 (or -3/16). This implies that three out of the sixteen games they played were low scoring. However, this value alone should not be used as the parameter as, since it is an extremely low value, the expected score calculation would change drastically. Therefore, to accurately represent the anomalies when computing the expected score, the value generated for each team would be added to their initial parameter of 1.0. This would mean that the finalized anomaly parameter for the Denver Broncos, for example, would be 0.8125.

```
Function updateAnomalyParameters(data, anomalies, anomaly_parameters)

        Create empty list representing anomaly teams
        Create empty list representing adjusted parameters

        For each anomaly in the anomalies
                Add both teams to anomaly teams

        Remove duplicates in anomaly teams

        Create empty dictionary representing number of games per team
        For each team in anomaly_teams:
                Set number of games to 0

        For each row in data
                For each team in anomaly_teams
                        If team equals value in 'Team1' column
                                Increment number of games by 1

         For each row in data
                For each team in anomaly_teams
                        If team equals value in 'Team1' column and team1 score 0 or < 7
                                Add (team, -1/number of games for team) to adjusted parameters
                        If team equals value in 'Team1' column and team1 score > 49
                                Add (team, 1/number of games for team) to adjusted parameters


         Create empty dictionary representing total parameters
         For each team in anomaly_teams
                Set total adjustments to 0

         For each pair in adjust_params
                Increment value of total_adjustments by value of parameter
                Set value for anomaly_parameters to 1 + total_adjustment

Return anomaly parameters
```

Figure 20: Pseudocode for the *updateAnomalyParameters* method

This new parameter is applied when calculating the expected points in the gamePrediction function. Initially, the expected points were calculated by taking the product of a team's offensive strength, their opponent's defensive strength, and the team's average points scored. The anomaly parameter can be included here by multiplying that expression by the team's anomaly parameter. This approach ensures that only minor changes in the expected score calculation occur based on the anomaly parameter. The expected score for Team 1, for example can be calculated as: (Team 1 Offensive Strength * Team 2 Defensive Strength * Team 1 Average Points Scored) * Team 1 Anomaly Parameter. In the case of the Denver Broncos in 2019, multiplying their expected score value by 0.8125 results in a slight reduction from their initial expected score.

4.6.1 2018 and 2019 Season Experiments

After running on the 2018 training set, the accuracy rate generated by the model was 67.38%. For the corresponding test set, the model returned an accuracy rate of 63.64%.

Regarding the 2019 training set, the accuracy rate reached 59.96% and, surprisingly, the accuracy rate for the 2019 test set was an astonishing 90.91%.

In comparison to the accuracy rates generated from the initial Poisson Distribution Model, with the exception of the 2019 test set rate, these new rates are lower across the remaining three sets. The minor fluctuations of the predicted scores as a result of the anomaly parameter implementation seemed to have switched the outcomes for several matches which has resulted in a lower accuracy rate. This may indicate that there are several close games in these datasets, where the winning team only narrowly defeats their opponent. Furthermore, these results showed that the model was better off without the implementation of this new parameter.

5 Decision Tree Experiment

5.1 Overview

Though the Poisson Distribution model accumulated positive results, it is insufficient to solely use that model to conclude that machine learning models can be used to predict American Football scores. To affirm this notion, it is necessary to implement a second machine learning model to see if it can also create accurate predictions. Therefore, a Decision Tree Regressor was implemented in order to see if other machine learning models, outside of the Poisson Distribution, can be used to predict American Football scores.

5.2 Code References

The inspiration for the code written for the Decision Tree Regressor algorithm is from the articles written by Sreejith P [22] and "The Click Reader" [9]. Furthermore, the inspiration for the code written to acquire the hyperparameters for the Decision Tree Regressor is from an article written by Nitin Kendre [23]. These articles helped foster ideas for implementation techniques for my Decision Tree Experiment.

5.3 Implementation

The first step to implementing the Decision Tree Regressor for this project was to define a feature set and a target variable. Since the feature sets are going to be used to calculate the target variable [9], it was necessary to re-calculate the offensive and defensive strength parameters as well as the average points scored by each team. This is to ensure that both the Poisson Distribution model as well as the Decision Tree Regressor take the same variables into account to make their estimations on the score. Once the parameters were re-calculated, using the mathematical computation approach, the next step was to figure out a way to incorporate them into the feature set. When examining other implementations of the Decision Tree Regressor, most feature sets were designed through extracting the desired variables from the dataset [9, 22]. Therefore, it was vital to add these parameters into the dataset to incorporate them, along with other variables, into the feature set. Along with these parameters, the feature set should also consist of the team names of both teams in the game. This would allow scores to be dependent on which teams are playing. However, taking the team names as presently constructed does not work as they are represented as strings. In her article, where she discusses encoding methods for data preprocessing, Shivani Singh states that "since the majority of machine learning models operate on numerical data,

categorical variables must be transformed into numeric form" [24], meaning that, to incorporate team names in the feature set, they must be translated into numerical values. This can be done through the implementation of a LabelEncoder.

```
Create instance of LabelEncoder
Call fit_transform function on teams in 'Team1', set values in new column 'Team1 Encoded'
Call fit_transform function on teams in 'Team2', set values in new column 'Team2 Encoded'
```

Figure 21: Pseudocode for the LabelEncoder usage

Once the values of the team names are encoded, they can be added into the dataset in a new column, and, since all the feature variables are now in the dataset, the feature set can be built.

In order to predict the score for both teams in a game, two target variables were created. One target variable was made for "T1 Score", and another for "T2 Score".

The next step is to construct the training and test sets for the model. In the Poisson implementations, the training and test were pre-defined as the Regular Season and Playoffs for a given season. However, for the Decision Tree algorithm, this approach would not suffice. Construction of the training and test set involves randomly splitting the training and test set based on a test size [9, 22]. Though the exact games cannot be represented in the test set, we can approximate the size of the test set to the same size as in the Poisson model. In the Poisson model, the test set accounts for approximately 22 rows while the training set accounts for 512 rows. The size of the test set can be expressed as 22/534, which evaluates to approximately 4.1%. The split also incorporates the usage of the *random_state* field. This field is an integral part of the code as it is used to "ensure that the results of the model are reproducible" [22]. The integration of this field guarantees that the output will remain the same through all iterations of the code.

With the training and test sets defined, the next phase of the implementation process was to tune the hyperparameters of the model. Hyperparameters are used in order to "control aspects of the learning process" [25] for a given algorithm and are beneficial for a model because of the fact that they can improve accuracy when making predictions [25]. Hyperparameters can be tuned through processes such as a *GridSearchCV*. The *GridSearchCV* is an algorithm that searches for the most optimal hyperparameters by "systematically exploring various combinations of

specified hyperparameters" [26] that suit the model. This allows developers to identify the best parameters that fit their model. In the case of this Decision Tree Regressor experiment, the two hyperparameters to fixate on are the maximum depth and minimum samples per leaf. The maximum depth parameter measures how deep the decision tree should grow [25]. The minimum samples per leaf parameter can be used to measure "the minimum number of samples that are required to be at a leaf node" [27]. The significance of using both these parameters together is that they prevent the tree from overfitting [27], in the case of minimum samples per leaf, and can analyse more intricate and complex patterns [25], in the case of maximum depth.

Since two target variables were used, to predict T1 Score and T2 Score, the *GridSearchCV* must be fit on both the training target variables.

```
Create parameter grid with values 'max_depth' and 'min_samples_leaf'

Create instance of GridSearchCV
Call fit function on training X and first training Y
Call best_params_, set returned values in bestparams1

Call fit function on training X and second training Y
Call best_params_, set returned values in bestparams2
```

Figure 22: Pseudocode for hyperparameter tuning implementation

Code inspired by [23]

The parameters returned by this code indicate that the ideal maximum depth for both trees was 3. However, the minimum samples per leaf differs for each instance. When training on the first target variable (*y1_train*), the minimum samples per leaf is 8, but, when training on the second target variable (*y2_train*), the minimum samples per leaf changes to 10. The change in the second minimum samples per leaf value could negatively impact the results of the experiment as the larger value could "prevent the tree from learning the data" [28].

After securing the parameters, the next step is to create an instance of a Decision Tree Regressor and fit them on the training variables [9, 22]. Since this experiment utilizes two target variables, two instances of a Decision Tree must be created. One of the Decision Trees must be fit to the first training target variable, and the other Decision Tree must be fit to the

second training target variable. Once the Decision Trees are fit to each target variable, predictions can then be made.

To measure the accuracy of this model on the data, it was necessary to add the predicted scores for each team back into the dataset to compare between the differences between a team's actual score and their predicted score. A similar function, used for evaluating the model in the Poisson Distribution experiments, can be used here to calculate the accuracy rate. The evaluation function can be further improved by calculating the average difference between the predicted score generated by the model and the actual score. This will provide insight as to whether the model is making accurate predictions regarding the actual score values.

5.4 2018 and 2019 Season Experiments

When this algorithm is run on the complete dataset of 2018 scores, consisting of both the regular season and playoffs, it returns a 50.0% accuracy rate on the test set. In addition to this, the model also states that the average difference between the predicted and actual scores for teams in the 'Team1' column is approximately 4.57 and approximately 0.36 for teams in the 'Team2' column.

On the dataset of 2019 scores, the model nets an accuracy rate of 52.27% on the test set. Regarding the average point differences, the model generated a difference of approximately 1.06 for teams in the 'Team1' column and 2.86 for teams in the 'Team2' column.

Overall, the results from this experiment show that a Decision Tree Regressor should not be the algorithm of choice for predicting the outcome of games considering that the accuracy rates are significantly lower than those from the Poisson Distribution model. However, this experiment did prove that Decision Tree algorithms can predict numerical values fairly close to their true value. The fact that the average differences between the actual and predicted scores are within 5 points proves that the algorithm had success in closely estimating the numerical value of the score.

6 Comparison with Betting Odds

6.1 Overview

The use of betting odds in this project serves as a practical prediction metric for the model. Betting odds can be used as a predictor for the outcome of a given match. For example, if the odds make one team the favorite, that implies that they are predicting that team will win the game. In relation to this project, the aim was to further examine the capabilities of machine learning models by seeing if they can also outperform predictions of bookmaker odds and be used to assist bettors in placing more accurate bets. This section of the project fixates more on a comparison with the outputs from the Poisson model, that utilized mathematically calculated parameters, because it can be used to calculate overall win probability.

The odds metric used in this project is a spread. In sports betting, a spread is used to identify whether a team is "expected to win or lose by a certain margin" [29]. A spread can be utilized to interpret who an oddsmaker is forecasting to be the favorite, with a negative spread [30], and the underdog, with a positive spread [30].

In order to visualize the meaning of a spread, an example can be created. In his article, Dan Preciado describes a game between the Buffalo Bills and the Miami Dolphins in which he states that the Bills have a spread of -7 and the Dolphins have a spread of +7 [29]. This implies that the Bills are expected to win against the Miami Dolphins by a margin of seven points [29]. On the other hand, the Miami Dolphins are considered to be a "seven-point underdog" [29] and are predicted to lose the match by seven points. This example articulates how a positive and negative spread can be interpreted.

The spread itself, however, is not enough to compare with the results from the models. To have an exact comparison with the outputs generated from the machine learning models, it is essential to convert the spreads into win probability percentages to replicate the type of probabilities generated from the Poisson model. In his article [31], Jimmy Boyd outlines the procedure necessary to convert the spread into an expected win percentage and calculates them for both the predicted favorite as well as the underdog.

Spread	Favorite Expected Win %	Underdog Expected Win %
0	50	50

0.5	50	50
1	51.3	48.8
1.5	52.5	47.5
2	53.5	46.5
2.5	54.5	45.5
3	59.4	40.6

Figure 23: Table with examples of win percentages

Values sourced from [31]

The table above illustrates an example of some of the win percentages for the corresponding spread values. In his piece [31], Boyd calculates the probabilities for spreads up to 16.5. With the spread values in the database, along with a conversion to win probabilities, the implementation process could be initiated.

6.2 Implementation

To properly modify the existing implementation of the Poisson to compare against odds, the first step was to correctly map the spread value for each team in a given match to its corresponding percentage using a new function. This part was quite extensive given the need to check for all cases in which Team 1 was the favorite or underdog and where Team 2 was the favorite or underdog. Due to this, the code within this function was repetitive, but thorough as all cases were accounted for.

```
Function transformOddsToProbabilities(team1, team2, data)

    Set team1 odds variable to 0
    For each row in data
        If team1 equals row's 'Team1' and team2 equals row's 'Team2'
            Set team1 odds to row's T1 Spread

    Set team2 odds variable to 0
    For each row in data
        If team1 equals row's 'Team2' and team2 equals row's 'Team1'
            Set team2 odds to row's T2 Spread

    Set team1 odds probability variable to 0
    Set team2 odds probability variable to 0

    #Checks for team1_odds and team2_odds values and maps to corresponding
    probabilities#

    #Example#
    If team1_odds equals 0 or -0.5
        Set team1_odds_probability to 50.0

    #Several rows follow for different values and probabilities#

Return team1 odds probability and team2 odds probability
```

Figure 24: Pseudocode for *transformOddsToProbability* function

Having the converted probabilities in hand, the next step was to modify the existing function used to evaluate the model to account for the odds probabilities as well. The intended purpose of changing this function was to measure how accurate the odds were, in predicting the winner, alongside the model. The procedure to do this was as follows. First, the betting favorite had to be accurately identified in each match. This could be found by examining which team had the higher calculated win probability. The following action was to see if the betting favorite for each match was the actual winner. If so, then this would be classified as a correct prediction for the odds. The accuracy rate could then be found by dividing the number of correct predictions by the total number of predictions, as calculated for the previous experiments.

```
#Existing implementation for evaluateModel function#

Set odds accuracy rate variable  to 0.0
Set correct odds prediction variable to 0
Set total odds prediction variable to 0

Create empty real winner variable
Create empty betting favorite variable
Create two empty lists representing all odd differentials for team1 and team2
Set total odd predictions variable to 0

For each row in dataset
        #Existing implementation for evaluateModel function#
        If row's 'Team2' and 'Team1' not in completed_matches
                #Existing implementation for evaluateModel function#
                #predicted_scores is returned earlier#

                #gamePrediction was modified to return odds probabilities and
                #differentials for each team
                Set odds probability 1 to predicted_scores[2]
                Set odds probability 2  to predicted_scores[3]


                #Existing implementation for evaluateModel function#

                If team1 score > team2 score
                        real winner = team1
                Otherwise
                        real winner = team2

                If odds probability 1 > odds probability 2
                        Set betting favorite to team1

                If odds probability2 > odds probability1
                        Set betting favorite to team2

                If betting favorite equals real winner
                        Add betting favorite to the accurate favorite selections
                        Increment total odd predictions by 1
                Otherwise
                        Increment total odd predictions by 1


                #Existing implementation for evaluateModel function#

                Calculate odds accuracy rate as len(accurate favorite selections)/total odd
                predictions * 100 rounded to the hundredths

Return string message with odds_accuracy_rate
#Along with other variables returned in function#
```

Figure 25: Pseudocode for odds accuracy code

6.3 2018 and 2019 Season Experiments

To further ensure that an equal comparison was being made with the previous experiments, it was imperative to test this implementation on the same datasets. The experimented betting odds were approximately 67.19% accurate on the 2018 Regular season and 54.55% accurate on the 2018 playoffs.

Regarding the 2019 data, the odds returned an accuracy rate of 64.06% on the training set and 63.64% accurate when predicting on the test set.

In comparison to the rates returned from the initial Poisson model, these rates are slightly less accurate. This is a very surprising outcome since the probabilities derived from the Poisson are based on only three parameters: the offensive and defensive strengths along with the average points scored by a team. On the other hand, it is implied that oddsmakers consider more factors when creating odds. Reportedly, oddsmakers take advantage of the fact that "computers can incorporate more data than any human ever could" [32]. Based on this statement, it can be deduced that oddsmakers must consider more than three variables when designing odds. As a result, the notion that the Poisson model with fewer input parameters is consistently outperforming odds generated from oddsmakers is astonishing. This experiment additionally concludes that, given the results from the model being more accurate than those from betting odds, individuals would be able to make more accurate bets using a machine learning model as opposed to relying solely on odds.

6.4 Further Testing

Since the accuracy of the odds were not up to par with the machine learning model, an additional step of this experiment was designed to see how far apart the probabilities from the odds were compared to the probabilities generated by the model. This was conducted by finding the average of the differences in probabilities for each team in a given match.

```
#Average difference calculation was done with modifications on both the gamePrediction
and evaluateModel functions#

#Modification of gamePrediction function#

Create empty list representing team1 odd differentials
Create empty list representing team2 odd differentials
Subtract team1 odds probability from team1 overall probability and add to odd
differentials1
Subtract team2 odds probability from team2 overall probability and add to odd
differentials2

Return odd_differentials1 and odd_differentials2
#Along with other variables returned in function#
```

```
#Modification of evaluateModel function#
Set variable avg_odd_differential to 0.0
Create empty list called all_odd_differentials1
Create empty list called all_odd_differentials2


For each row in dataset
        #Existing implementation for evaluateModel function#
        If row's 'Team2' and 'Team1' not in completed_matches
                #Existing implementation for evaluateModel function#
                #predicted_scores is returned earlier#

                Set odd differentials 1 to predicted_scores[4]
                Set odd differentials 2 to predicted_scores[5]

                #Existing implementation for evaluateModel function#

                For each differential in odd differentials1
                        Round differential to hundredths and add to all odd differentials1

                For each differential in odd_differentials2
                        Round differential to hundredths and add to all odd differentials2


                #Existing implementation for evaluateModel function#

                Calculate average odd differential as sum of all odd differential lists
                together divided by the length of both lists


Return string message with avg_odd_differentials
#Along with other variables returned in function#
```

Figure 26: Pseudocode for average difference calculation here

The objective of this was to see if the probabilities returned from the odds as well as the model are similar or very far apart. The average difference between the odds probabilities and the model generated probabilities was approximately 2.6% on the 2018 training set and 1.73% on the 2018 test set.

Additionally, on both the 2019 training and test set, the average difference between the two probabilities was 1.45% and 2.08%.

Though these averages imply that both probabilities are quite close to each other, this may not be the most accurate metric. When looking at the outputs of this experiment, there seem to be several games in which the model-generated probability is either significantly higher or lower than the probability from the odds. An example of this can be found in the 2019 test set, in a match between the Baltimore Ravens and the Tennessee Titans. Below is the output of that game. In this game, the model generated an expected win probability of 75.18% for the Tennessee Titans and 19.83% for the Baltimore Ravens. However, the odds favored Baltimore, with a converted win probability of 83.6%, and viewed Tennessee as the underdog, with a converted win probability of 16.4%. The differences between the odds derived probabilities and the model-generated probabilities resulted in -58.78 for Tennessee and 63.77 for Baltimore. As a result, it calls into question whether the low average difference value is a consequence of the wide dispersion of differentials. This can be verified

through computing the standard deviation of the differentials between the odds and model-generated probabilities. Standard deviation is a measurement that aims to see "how far individual points in a dataset are dispersed from the mean" [33]. When looking at the 2018 data, the standard deviation of the differentials between odds and the model-generated probabilities was approximately 28.05 for the training set and approximately 28.27 for the test set. Similarly, for 2019, the standard deviation was approximately 28.33 and 31.24 for the training and test set. These metrics directly support the claim that the odds differ significantly from the model-generated probabilities and that the low average difference is attributed to the wide distribution of differentials. Based on these results, it is apparent that the Poisson model and the odds have varying levels of confidence when computing the expected win percentage for each team as, in some instances, the model generates a significantly higher probability prediction and, in other cases, the odds have the higher probability prediction.

7 Conclusion

With the conclusion of the experiments, we can now evaluate the performance of each model, and bookmaker odds, to determine the most optimal method for predicting American Football scores.

Model	**2018 Training Set Accuracy**	**2018 Test Set Accuracy**	**2019 Training Set Accuracy**	**2019 Test Set Accuracy**	**Average Accuracy Rate**
Poisson (optimized parameter method – initial execution)	50.39%	59.09%	72.46%	81.82%	65.94%
Poisson (mathematically calculated parameter method)	67.97%	63.64%	67.77%	90.91%	72.57%
Dixon-Coles-Style Modification on Poisson	67.38%	63.64%	59.96%	90.91%	70.47%
Decision Tree Regressor	N/A	50.0%	N/A	52.27%	51.14%
Betting Odds	67.19%	54.55%	64.06%	63.64%	62.36%

Figure 27: Table with all results from each model

The table above illustrates the results across all of the experiments in this project. Since the accuracy of the Decision Tree Regressor was not measured on a training set, the values within the Training Set Accuracy fields, for both 2018 and 2019, were set to N/A. Also, as mentioned in Section 4.6, the Dixon-Coles inspired modifications were implemented on the Poisson model that utilized the mathematically calculated parameter approach because of its consistency through multiple executions of the algorithm.

In order to comprehensively evaluate the results of all models and determine the best performing algorithm, it was necessary to calculate an additional metric that could be used to fairly compare all five experiments. Though in the previous sections evaluations were made by comparing the performances across all four datasets with one other algorithm, this approach would not suffice in this case. When looking at the overall results, it is apparent that the most accurate model varies based on the dataset. For example, the Poisson model that utilized optimization methods had the highest accuracy rate on the 2019 training set. However, both the Poisson model, that utilized mathematically derived parameters, and the Poisson model with the Dixon-Coles style modification had the highest accuracy rates on the 2018 and 2019 test sets. As a result, counting the datasets where each model had the highest performance and adding up those occurrences to see which one had the most, would not result in an accurate representation of the most optimal model.

To correctly identify the best performing model, it is possible to use the average accuracy rate returned across all datasets in the experiment. The average accuracy rate provides an overall representation of how the model performed on all of the datasets that were experimented on. This can be computed by summing the accuracy rates and dividing by the number of datasets experimented on for each model. After computing the average accuracy rate for each model, the models can be ranked based on which one netted the highest average rate.

Overall Model Rankings:

1. Poisson model (mathematically calculated parameter method): 72.57%
2. Dixon-Coles Style Modification on the Poisson model: 70.47%
3. Poisson model (optimized parameter method): 65.94%
4. Betting Odds: 62.36%
5. Decision Tree Regressor: 51.14%

The model that was the least accurate overall, and therefore the least optimal choice for predicting American Football scores, was the Decision Tree Regressor. Since the accuracy of the Decision Tree Regressor was not computed on the training sets for either 2018 or 2019, the performance of the model is being measured on a significantly smaller sample size compared to the other approaches. Though this could be part of the reason for the model's results, it is important to note that even on its experimented test sets, for the 2018 and 2019 data, the Decision Tree Regressor was consistently the worst performing algorithm.

The model that was the most accurate overall, and therefore the most optimal choice for predicting American Football scores was the Poisson model with the parameter calculation approach. This approach consistently netted high accuracy rates relative to the other models. Furthermore, the three best performing algorithms are variations of the Poisson Distribution model. This proves that any variation of the Poisson model can be used to accurately predict sports scores. Further applications of this project could involve comparing a Poisson Distribution against another machine learning model, such as a Random Forest, in order to investigate if a Poisson Distribution is the most optimal model to use in general. However, in the context of this project, it can be concluded that the Poisson Distribution model, that utilized mathematically derived parameters, is the most accurate choice for predicting American Football scores.

Given that it has been determined that the Poisson model can be used to accurately predict American Football scores, an additional goal for this project was to see if machine learning models can be used to place more accurate bets. To answer this, we can utilize the average accuracy rate derived earlier and compare its values for each model in relation to the betting odds. The results above indicate that the three variations of the Poisson model outperform the predictions of betting odds. However, it is important to note that the Poisson model, that utilized parameter optimization, may not always generate more accurate predictions than betting odds. As outlined in Section 4.4.3, that variation of the model will generate different outcomes in each execution of the code. As a result, it cannot be expected that this model will consistently outperform betting odds, which means that only two variations of the Poisson model can be used to place more accurate wagers:

1. Poisson model with mathematically derived parameters
2. Dixon-Coles Inspired modification on the Poisson model

In conclusion, this project provides definitive proof that, though not perfectly, machine learning models can be used to approximate scores from sports other than Association Football and can be used to make slightly more accurate wagers. The findings in this project also illustrate that machine learning models can generate results that outperform those from various sources, such as oddsmakers.

In the initial stages, the goal of this project was to experiment to see how machine learning models would fare, in comparison to the implementations for Association Football as well as experts like oddsmakers, in predicting the outcomes for American Football fixtures, and, when reflecting on these experiments, there are further improvements

that could be made to this project to reaffirm the outlined motives. The most significant potential improvement to this project would be to account for more variables in an American Football game. In a match, there are other factors that can influence the outcome, besides the defined parameters in the experiment, such as changes in a team's roster. It is not certain that a team will maintain the same roster throughout the course of an entire season due to injuries or trades, for example. In this experiment, the parameters were derived through an evaluation of a team's performance. However, in order to account for potential roster changes, an improvement to this project could be to create a valuation metric for each offensive and defensive player and use those in order to construct the offensive and defensive strength parameters. This would mean that, if a player is no longer on a team, their strength parameter would be affected based on how valuable that player was for their team.

An additional improvement on this project would be through the comparison of human crafted predictions, in the betting odds experiment, as opposed to a betting spread, which is influenced by "contemporary computing power" [32]. These could consist of predictions made by individuals within sports media or broadcasters, for example. Seeing how machine learning models fared when predicting outcomes against betting odds, the use of human crafted predictions would allow for the direct comparison between machine learning models and humans.

7.1 Self-Assessment

Creating this project was a very rewarding experience as it allowed me to gain experience in managing a large project. One of the most significant takeaways from working on this project was the importance of time management. The nature of this project involved an examination of numerous resources, the creation of datasets, multiple coding experiments, and an analysis of the work. As a result, effectively managing time throughout each process allowed me to complete each sector of the project in a thorough and efficient manner. The technical portions of this project also turned out well. Through research, I was able to learn about the intricate workings of many models from first principles, which helped strengthen my understanding about machine learning. This project also allowed me to gain experience in experiment development. Being able to take an idea, analyse existing research and figure out how to extend it to implement the idea, will serve me well in industry. Additionally, my familiarity with Python aided me in coding efficiently.

Though the project, for the most part, went according to plan, there were aspects in the process that could have been improved. One such aspect

was the data collection for the datasets. As mentioned in the Data Collection section, this was done through manual scraping. The issue with manually scraping the data was that it was very time consuming. Utilizing an API or an existing scraping algorithm would have made completing this part of the project faster. This speaks to the importance of time management in a large project like this. Completing a simpler task like this in a shorter time would have allowed me to start on other features of this project in a faster time.

I believe there could be several applications for this project beyond the sports industry. Though these experiments were primarily fixated on sports, the primary principle of this endeavour, which is the predictive power of machine learning algorithms, can be applied to a variety of industries. Next, I hope to conduct similar experiments in another industry that I am passionate about in order to attempt to solve more pressing manners in society with the use of machine learning models.

Bibliography

[1] M. Ebejer, "Poisson Distribution in Sports Betting [Step-By-Step Guide]🥇🥇," *ThePuntersPage.com*. https://www.thepunterspage.com/poisson-distribution-betting/

[2] "History of Sports Betting," Sep. 06, 2023. https://rue.ee/blog/history-of-sports-betting/

[3] C. Osorio, "How Sports Betting Impacts The Economy," *Money Digest*, Apr. 29, 2024. https://www.moneydigest.com/1570985/how-sports-betting-impacts-economy/

[4] G. R. Poleo, "Americans have lost $245BILLION on sports betting since 2018," *Mail Online*, Aug. 11, 2023. https://www.dailymail.co.uk/sport/nfl/article-12397551/Americans-lost-staggering-245BILLION-sports-betting-restrictions-loosened-2018-experts-fearing-gambling-addiction-gripped-nation.html

[5] E. Petrova, "What Exactly Is Poisson Distribution? An Expert Explains," *careerfoundry.com*. https://careerfoundry.com/en/blog/data-analytics/what-is-poisson-distribution/

[6] D. Sheehan, "Predicting Football Results With Statistical Modelling: Dixon-Coles and Time-Weighting," *dashee87.github.io*, Sep. 13, 2018. https://dashee87.github.io/football/python/predicting-football-results-with-statistical-modelling-dixon-coles-and-time-weighting/

[7] API FOOTBALL, "Mastering Scores in Sports: The Power of Poisson Distribution - API-FOOTBALL," *api-football*, Feb. 06, 2024. https://www.api-football.com/news/post/mastering-scores-in-sports-the-power-of-poisson-distribution

[8] Viswa, "Unveiling Decision Tree Regression: Exploring its Principles, Implementation," *Medium*, Jul. 31, 2023. https://medium.com/@vk.viswa/unveiling-decision-tree-regression-exploring-its-principles-implementation-beb882d756c6

[9] T. C. Reader, "Decision Tree Regression Explained with Implementation in Python," *Medium*, Oct. 19, 2021. https://medium.com/@theclickreader/decision-tree-regression-explained-with-implementation-in-python-1e6e48aa7a47

- As mentioned in the report, this article inspired the implementation for the Decision Tree Regression algorithm

[10] "Decision Tree," *CORP-MIDS1 (MDS)*. https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/

[11] A. Sharma, "Decision Tree vs. Random Forest - Which Algorithm Should you Use?," *Analytics Vidhya*, Jul. 29, 2024. https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

[12] "How does soccer scoring work?," *keepthescore.com*. https://keepthescore.com/blog/posts/soccer-scoring/

[13] "Scoring Plays | NFL Football Operations," *operations.nfl.com*. https://operations.nfl.com/the-rules/nfl-video-rulebook/scoring-plays/

[14] "Common Football Scorelines - What's the Most Frequent FT Scores? | FootyStats," *footystats.org*. https://footystats.org/stats/common-score

[15] "NFL Scores History: Highest/Lowest Scoring Game, Common Scores, and More," Dec. 10, 2023. https://www.profootballnetwork.com/nfl-scores-history-highest-lowest-scoring-game-common-scores-more/

[16] "SportsOddsHistory.com | Archived futures lines and outright markets," *SportsOddsHistory.com | Archived futures lines of the Super Bowl, World Series & more*, Apr. 26, 2014. https://www.sportsoddshistory.com/

[17] "pena.lt/y," *pena.lt*. https://pena.lt/y/2021/06/18/predicting-football-results-using-the-poisson-distribution/

- As mentioned in the report, the log_likelihood, fit_poisson, and fit functions from the Popular Poisson Experiments were sourced from this article
- Also, as mentioned in the report, the first few lines of the gamePrediction function (up until the plotting code) were inspired by this article

[18] "How to calculate Poisson distribution for football betting," *Smarkets Help Centre*, 2017. https://help.smarkets.com/hc/en-gb/articles/115001457989-How-to-calculate-Poisson-distribution-for-football-betting

[19] Eric, "Beginner's Guide To Maximum Likelihood Estimation," *Aptech*, Sep. 21, 2020. https://www.aptech.com/blog/beginners-guide-to-maximum-likelihood-estimation-in-gauss/

[20] "2018 NFL Regular Season Schedule Released | Pro Football Hall of Fame," *pfhof*, 2018. https://www.profootballhof.com/news/2018/04/2018-nfl-regular-season-schedule-released

[21] “How Many Games Are There In A Premier League Season (And How It Compares To Other Leagues),” *Jobs In Football*, Dec. 01, 2023. https://jobsinfootball.com/blog/how-many-games-in-a-premier-league-season/

[22] S. P, “Decision Tree Regression: With Python Code,” *Medium*, Feb. 24, 2023. https://medium.com/@sreejithp_30057/decision-tree-regression-with-python-code-e0d79f6ef81b

- As mentioned in the report, this article inspired the implementation for the Decision Tree Regression algorithm

[23] Nitin Kendre, “Decision Tree Regression : A Comprehensive Guide with Python Code Examples and Hyperparameter Tuning,” *DEV Community*, Jun. 05, 2023. https://dev.to/newbie_coder/decision-tree-regression-a-comprehensive-guide-with-python-code-examples-and-hyperparameter-tuning-1f0f

- As mentioned in the report, this article inspired the implementation of hyperparameter tuning in the Decision Tree Experiments

[24] S. Singh, “Title: Label Encoding and One-Hot Encoding for Data Preprocessing,” *www.linkedin.com*, Jul. 19, 2023. https://www.linkedin.com/pulse/title-label-encoding-one-hot-data-preprocessing-shivani-singh

[25] “How to tune a Decision Tree in Hyperparameter tuning,” *GeeksforGeeks*, Apr. 16, 2024. https://www.geeksforgeeks.org/how-to-tune-a-decision-tree-in-hyperparameter-tuning/

[26] R. Shah, “GridSearchCV | Tune Hyperparameters with GridSearchCV,” *Analytics Vidhya*, Aug. 13, 2024. https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/

[27] om pramod, “Decision Trees,” *Medium*, Jan. 29, 2023. https://medium.com/@ompramod9921/decision-trees-8e2391f93fa7

[28] scikit-learn, “1.10. Decision Trees — scikit-learn 0.22 documentation,” *Scikit-learn.org*, 2009. https://scikit-learn.org/stable/modules/tree.html

[29] D. Preciado, “What Is A Spread In Sports Betting?,” *Forbes*, Nov. 02, 2023. Available: https://www.forbes.com/betting/guide/what-is-a-spread/

[30] “Point spread, over/under, moneyline: NFL odds explained,” *FOX Sports*. https://www.foxsports.com/stories/nfl/point-spread-over-under

[31] J. Boyd, "Conversion Chart for NFL Point Spreads to Percentages or Money Lines," *Boyd's Bets*, Sep. 10, 2023. https://www.boydsbets.com/nfl-spread-to-moneyline-conversion/

[32] P. Cwiklinski, "How Do Bookmakers Generate Sports Odds?," *Sports Betting Dime*. https://www.sportsbettingdime.com/guides/betting-101/how-bookmakers-generate-odds/

[33] M. Hargrave, "Standard deviation formula and uses vs. variance," *Investopedia*, 2023. https://www.investopedia.com/terms/s/standarddeviation.asp

Coding References

This section contains a list of references used when coding the experiments for this project that are not explicity discussed in the report

[34] "SciPy — SciPy v1.5.4 Reference Guide," *docs.scipy.org*. https://docs.scipy.org/doc/scipy/reference/index.html

- This reference represents the SciPy library, which was used in this project. For example, the built-in Poisson Distribution and the minimize function are from this library

[35] "API Reference — Matplotlib 3.5.0 documentation," *matplotlib.org*. https://matplotlib.org/stable/api/index

- This reference represents the matplotlib library. In the experiments, this library was used when writing the code for plotting the Poisson Distribution for each team in a match

[36] "NumPy Reference — NumPy v1.19 Manual," *numpy.org*. https://numpy.org/doc/stable/reference/index.html

- This reference represents the NumPy library. In the project, the NumPy library was used for its built-in standard deviation and logarithmic functions

[37] omelias and pigrammer, "How to merge two dictionaries?," *Stack Overflow*, 2022. https://stackoverflow.com/questions/73660795/how-to-merge-two-dictionaries

- This reference is of a technique used to combine dictionaries. In the Dixon Coles inspired experiment, this technique was used to combine the anomaly parameter dictionary with the offensive and defensive strength dictionaries

[38] "API reference — pandas 1.1.4 documentation," *pandas.pydata.org*. https://pandas.pydata.org/docs/reference/index.html

- This reference represents the Pandas library. Functions within the Pandas library were used throughout this project such as read_csv, .concat, .iloc, .at, and more

[39] "API Reference," *scikit-learn*. https://scikit-learn.org/stable/api/index.html

- This reference represents the scikit-learn library. The Decision Tree experiments utilized instances of the DecisionTreeRegressor, LabelEncoder, and GridSearchCV classes as well as the built-in train_test_split function

[40] "python - How to deal with SettingWithCopyWarning in Pandas," *Stack Overflow*. https://stackoverflow.com/questions/20625582/how-to-deal-with-settingwithcopywarning-in-pandas

- This reference provides insight as to how to fix a SettingWithCopyWarning. This was used in the Decision Tree experiments to prevent these warnings from persisting. The line of code referenced from this source was 'pd.options.mode.chained_assignment = None'.

How to Download and Use this Project

Pre-Requisite for Running the Project:

- Must have installed Python 3.x
- Must have installed Jupyter Notebook
- Must have installed the following libraries:
    - Pandas
    - SciPy
    - Matplotlib
    - NumPy
    - Scikit-Learn

Use these links for instructions on installing the libraries if needed:

- Pandas: https://pandas.pydata.org/docs/getting_started/install.html
- SciPy: https://scipy.org/install/
- Matplotlib: https://matplotlib.org/stable/install/index.html
- NumPy: https://numpy.org/install/
- Scikit-Learn: https://scikit-learn.org/stable/install.html

If you do not have Python 3 or Jupyter Notebook Installed:

- Google search 'How to install Python3' for your environment. Below are some helpful links that provide instructions for installing Python 3, as of the writing of this report, on various environments.
    - *Windows:* https://phoenixnap.com/kb/how-to-install-python-3-windows
    - *Mac:* https://programwithus.com/learn/python/install-python3-mac
    - *Linux:* https://www.geeksforgeeks.org/how-to-install-python-on-linux/
- Google search 'How to install Jupyter Notebook' for your environment. Below is a helpful link that provides instructions for installing Jupyter Notebook, as of the writing of this report.
    - https://jupyter.org/install

With Python 3.x and Jupyter Notebook installed, follow these instructions for running the experiments:

1. To run the experiments in this project, the first step is to unzip the submitted zip file into a directory
2. Once unzipped, you should see a folder called MyProject
3. Open the folder, and the following contents should be in the folder
    - Poisson_Experiment_2018 (Jupyter Notebook File)
    - Poisson_Experiment_2019 (Jupyter Notebook File)
    - DT_Experiment_2018 (Jupyter Notebook File)
    - DT_Experiment_2019 (Jupyter Notebook File)

    - Dixon_Coles_Experiment_2018 (Jupyter Notebook File)
    - Dixon_Coles_Experiment_2019 (Jupyter Notebook File)
    - Popular_Poisson_Model_2018 (Jupyter Notebook File)
    - Popular_Poisson_Model_2019 (Jupyter Notebook File)
    - Poisson_Odds_Experiment_2018 (Jupyter Notebook File)
    - Poisson_Odds_Experiment_2019 (Jupyter Notebook File)
    - 2018 Regular Season Data (CSV File)
    - 2018 Playoffs Data (CSV File)
    - 2019 Regular Season Data (CSV File)
    - 2019 Playoffs Data (CSV File)
    - Copy of this Report (PDF)
4. Ensure that the notebooks and the datasets remain in the same folder
5. Open new Terminal window
6. Change directory to the folder where the above files are located
    - In the terminal, you should be able to list and see all the files above using commands such as ls
7. Run Jupyter Notebook using the command: jupyter lab
8. In Jupyter Lab, double click on the notebook file you wish to run
9. Press Run Button
10. For the Poisson, Dixon Coles, Popular Poisson, and Poisson Odds experiments, which are each in separate notebook files, the execution protocol is as follows:
    - Run the first cell in each notebook in order to see the results on the training set
        - Ensure to click 'Click message to see full output' if prompted
    - Run the second cell in each notebook in order to see the results on the test set
11. For the Decision Tree experiments the execution protocol is as follows:
    - Run the first cell in each notebook in order to see the results of the experiment

Important factors to consider:

1. The Popular_Poisson_Model experiments, for both 2018 and 2019, take longer to run (approximately 30 minutes)
2. The experiments can be run in any order