Comparing Sports Prediction Models

Using Perceptrons, Naïve Bayes, and K-Nearest Neighbors

Nathaniel Greene
Computer Science
Binghamton University
Binghamton NY United States

ngreene5@binghamton.com

ABSTRACT

With the extreme popularity of professional sports, the social aspect of being a sports fan continues to grow. From gambling on scores to creating a strong fantasy sports team, insights and predictions on sports games are a very valuable commodity, the more accurate the better. Currently there are plenty of different sources that can give a prediction on a player or team's performance in a given match. The goal of this project was to test the accuracy of different machine learning techniques on publicly available databases of football games and observe how useful each one is at predicting the outcome of future games. These models are somewhat basic compared to the possibility of what can be created to predict sports games, however they serve as a solid foundation for what the ideal modeling system would look like in this field of predictions.

1 Introduction

The main question I look to solve (or begin solving) with this project is, "How accurately can I use statistics from completed NFL games to predict the outcomes of future games?" The main two important parts of beginning to accomplish this task is what is the ideal data source that will be the most useful, and what algorithm or methodology will be the best way to analyze this data. I see sports betting more prominently every week, and a lot of people will place a bet on what they *think* will happen, but it seems like a task that can be more successfully accomplished by a computer using more data than a human can process. There is always the possibility of something that can't be predicted in live sporting events, but unlike casino betting games that mathematically give an edge to the casino, it appears that there are ways to increase your odds on sports betting significantly enough to give a bettor an advantage.

1.1 Methodology

The three machine learning techniques I decided to use are perceptrons, Naïve Bayes, and the k-nearest neighbor method. The data following results of sporting events can be noisy sometimes, so the algorithms used here may not be able to perfectly classify even the training data once training is complete, but I need to find the most accurate parameters for each. The perceptron model took some guess-and-check work to determine what the optimal learning rate and bias were with a large number of loops, and ended up being most effective with a relatively small learning rate and a larger negative bias. With the Naïve Bayes model, all training data was used for a per-team average on all features, and then ran against the test data. The k-nearest neighbor (KNN) algorithm stores all of the sample data, and computes the distance from each test game. I tested a variety of different k-values, and chose the most accurate number of points that correctly predicts the highest number of game outcomes.

1.2 Data Sets

I chose to use the 2009-2019 NFL seasons for this project. My training data consists of all regular and post-season (playoff) games from 2009 to 2018, and the 2019 full season was used to test each algorithm. I did not choose 2020 because there is a factor that is difficult to account for in this dataset: the COVID-19 pandemic. I did not use the 2021 season because it is not complete yet and I wanted to test a complete season as the training data. I got my data from Pro Football Reference[1], which gave me a data set for each of my desired seasons, broken down into a .csv file for each season. The files have every game from the complete season from each week, as well as the playoff games. Each game has the week (1-17), date, time of day and day of week, which team was at home and which was away, as well as the points scored, yards gained, and turnovers from each team involved. When taking in this data to my code, I parsed each game into a list for both teams involved and converted non-numerical values (such as day of the week) into a format that could be used for the machine learning algorithms. I parsed the test data games in a similar fashion, although I had to change a few values. Since this is the test data, I would normally not know the outcome of the games because I would be attempting to predict what will happen before the game occurs. For this, I used all of the training data for each team to give each team in the game a score, gained yards, and turnover statistic based on that teams average for each category. This design creates inaccuracies when running the test data, but overall gave more value to these specific features rather than ignoring them completely when training/testing. In the test set, the 2019 NFL complete season, there were 267 games played, and for each algorithm used the total number of correct predictions was compared to this total to compute accuracy. Another key aspect to note is that for each game, two teams compete, so if one wins the other team must lose. Since the data is based on real world events, it is not perfect, so when running each algorithm I computed the prediction for *both* teams involved in each game. In some cases, team A would expect a win and team B would expect loss, so the output of these examples are trivial. However, if both teams expect the same outcome, I can compare the accuracies of the model for each team individual, and then select which result to use based on the more accurate model.

2 Results

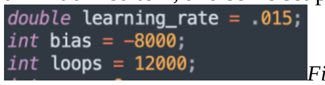
The three algorithms I decided to use all showed around the same accuracy level on this test data. The models were consistently accurate more than half the time, which is a great start when it comes to betting. If a bettor was to follow these results for each season game in 2019 and bet on every single game, he or she would bet correctly more than half the time, and this points towards profit.

2.1 Preparing The Data

Before any of the three algorithms can run, the training and test data have to be parsed in and made useful to the code. The training data is read in file by file, and each game is added to two lists, one for each of the teams involved in that game. Once the training data is completely read into the code, each team has their own list containing every game they were involved in over the years 2009-2018. The test data is parsed in a similar way, but stored differently. Each game is added twice to a single list, once for each of the perspectives of the teams involved. The team that the statistics in each entry of this list pertain to is also stored to allow the algorithms to find the corresponding teams trained model. This feature is ignored when it comes to testing as it is only to signify which team's statistics are being used.

2.2 Perceptrons

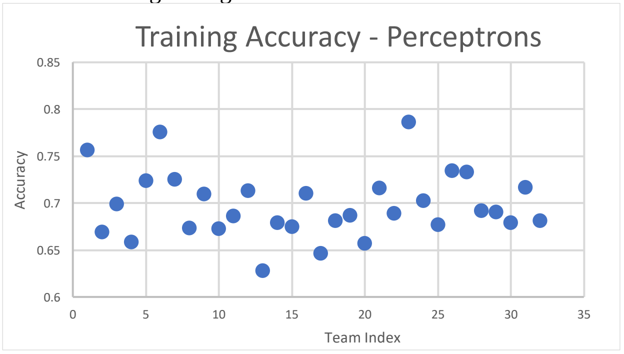
The first model tested was the perceptron model, where each feature is given a weight, and the training data is looped through numerous times, each time updating the weights based on a set learning rate, the feature value, and whether or not the prediction with the current weights is correct. Once the initial training data sets are parsed into the code, the perceptron training is ran for every team. In each loop, the given team is given a list of weights, all initialized to 1, and some set parameters (Figure 1).



```
double learning_rate = .015;
int bias = -8000;
int loops = 12000;
```

Figure 1.

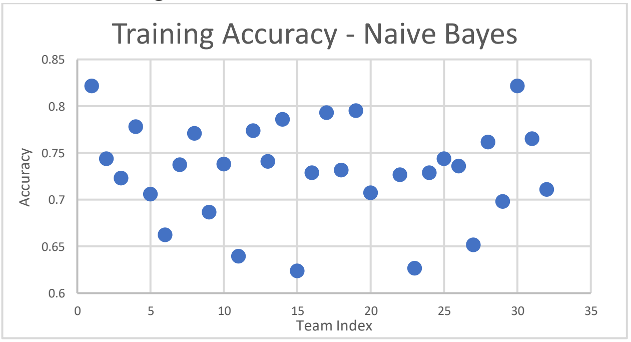
After testing, these were the values of learning rate, bias, and number of cycles that made the perceptron model the most accurate. I began with larger learning rates, which were initially inaccurate, and this led me towards smaller learning rates to find more accuracy. I then tested the bias, and found that a numerically large negative bias allowed for the most accurate results and decreased the number of loops to maintain this accuracy. I then had to test the loops to find the least amount of cycles before the accuracy did not increase and the results became redundant but caused the program to run longer. The target number of loops causes the program to run much slower than it would with less loops, making the perceptron model the longest in runtime of the three. On the final iteration of the perceptron training, accuracies with the weights against all of the test was recorded.



After training, each team's weight vector had between a 63% and 78% accuracy on the training set. Once each team has its own list of accurate weights stored, the model is tested. Two entries from the test list are passed in at a time, one for each team involved in the game. The first team passed in is the winning team and the second is the losing, and for each team a total value is calculated from the sum of each weight times the corresponding feature value. For each list of weights, the last value is the accuracy on the training data, and these values are compared to determine which team's output to use as the final output. If the first team is chosen and is predicted a win, that is correct, and if the second team is chosen and predicted a loss, that is correct, and a counter is increased by 1. Otherwise, the prediction is wrong, and the counter for correct predictions remains the same. With this method, 158 out of the 267 were predicted correctly, which is approximately 59.18%.

2.3 Naïve Bayes

Similar to the perceptron algorithm, the Naïve Bayes model loops through each team's list of games. It begins by calculating the mean and variance for each feature for the given team based on all training examples. These values are stored in respective lists, and globally saved so they can be accessed when testing begins. A note about the mean and variance lists is that they ignore the index of the two teams involved in the game because these are not necessarily statistics relating to the outcome of the game, but rather arbitrary reference values I gave. After the mean and variance is calculated and stored for each feature for each team, the training data is ran through using the values discovered. Below is the accuracy for each team predicting their own games from the training class.



This chart has the same scale as the chart displaying perceptron training accuracy. In the Naïve Bayes training tests, accuracies for each team are on average slightly higher than the perceptron model, with more variance in the points. After training data has been tested, the test data is ran through a probability density function.

$$p(feature|class) = \left(\frac{1}{\sqrt{2}(\pi)(\sigma^2)}\right)^{\frac{-(f-\mu)^2}{2(\sigma^2)}}$$

In this equation, σ^2 refers to variance of the feature, μ is the mean, and f is the current feature's value in the test data. Log scaling is also used to make the numbers more manageable, as they are very large after using this equation. Similar to the perceptron, this also uses two entries from the parsed test data per loop for each team involved. Here, a teams probability of winning and losing are compared, and whichever value is greater, that option is chosen. Since there are two teams, the highest value among both teams is chosen in order to use the highest confidence model. Again, out of 267, the Naïve Bayes model correctly predicted 153, giving it an accuracy of approximately 57.30%, slightly underperforming the perceptron model.

2.4 k-Nearest Neighbors

The final method of prediction used was the k-nearest neighbor (KNN) structure. A KNN model allows for quicker training time because during training it only needs to store each training data point, no calculations required. Since the training data is parsed and stored earlier in the code when the perceptron model trains, the training phase for KNN is already complete. When testing the KNN model, similar to choosing parameters for the perceptron model, *k* must be varied while accuracy is measured. After testing values of k from 1 to 101 (odd numbers were used to avoid ties), the most effective *k* was measured at *k*=47. Below is some benchmarks to show correlation between *k* and how accurately it classifies the test data.

k	#correct	#total	accuracy
1	24	267	0.08988764
5	89	267	0.33333333
15	132	267	0.49438202
25	144	267	0.53932584
35	152	267	0.56928839
45	152	267	0.56928839
47	161	267	0.60299625
49	151	267	0.56554307
51	146	267	0.54681648

The reason accuracy is so low when there is a small value of *k* is because of ties. The odd value of *k* avoids causing the number of wins and losses to tie, but since I also compare the higher value between both teams, ties occur more frequently when looking at less nearby points. If *k*=1, and both teams end up predicting a win based on the single nearest point, this is a tie. In my testing, I count a tie as incorrect, because with this tie I would leave it up to random chance of which team is used. Random selection would increase accuracy, but the mathematical worst case is choosing the wrong team every time, so I went with this to calculate the optimized real accuracy. When the KNN is tested, each game in the test season is ran 2 at a time, the same way as the previous 2 models. For each team, the distance is calculated from each point in that team's set of training data to the current point in the test set. The point is then stored in a list, added in ascending order, and the corresponding index in the test set is added at the same location in a separate list. Once all distances have been calculated and stored from smallest to largest, the smallest *k* number of examples are taken and the total wins and losses are calculated based on these (*k*=47) number of closest points. The purpose of storing the indexes in another list is so the model can check what the correct output of each point is from the training example. Between the number of losses and wins, whichever number is largest, that is the class this test example is labeled with. Between the two teams involved, whichever has a greater maximum number is the team that is used to determine the outcome of the game. This is the most accurate of the three models, correctly predicting 161 games out of 267, giving an accuracy of around 60.03%.

3 Overview

All three models used in this work showed signs of effectivity in predicting the outcome of "future" NFL games based on past statistics. All three models came out with around a 60% success rate, which shows lots of room for improvement but still provides value in the fact that it is a significant amount over 50% when it comes to placing a bet.

3.1 Analysis of Results

Although the KNN was only a small margin more accurate than both the perceptron and Naïve Bayes models, I still believe it is the most effective method of classifying and predicting games. With the data used here, a handful of features are less mathematically based, such as the day and time of the game. Knowing numerical statistics of a game is certainly important, for example knowing the score each team gets in a game clearly shows who wins, but these are values one would not have *before* a game. However, the non-numerical based stats are available to use when predicting a game's outcome. Each feature needs to be transformed into a numerical representation for the models to work. With the first two, finding a proper weight or using the mean and variance of values such as day of the game will not always be helpful because they are somewhat arbitrary. When using the KNN, values like these don't need to be mathematically weighted, because the relationship between the test feature and trained feature is based on similarity. I believe it is important to take into account non-numerical feature relations because trends within this type of information have an impact on the outcome of the game. Beyond the scope of the data set I used, aspects such as weather or type of stadium can be useful information to include, and classification of these would work well with a KNN model. If a team loses 95% of the time when it is snowing, for example, many of the nearest neighbors to a test point in a snowy game would show a loss based on distance.

3.2 Improvements and Future Work

Through completion of my work, I noticed numerous ways I would be able to improve these models moving forward. One of the seemingly most important aspects to change would be the data set. The reason I chose to use the seasonal data is because it was the most well organized and inclusive of all teams and all games for each season. However, I think a strong step to increase accuracy would be to use a dataset of individual *player* statistics for each team. This would require custom modifications and construction of the desired dataset, which is why I did not use this format here, but taking the statistics of each player on a team to predict performance as a whole would be beneficial. If a specific player is injured before a game, or is traded to another team, the proper adjustments can be made before calculating a winner. Moving forward with the current team-based data sets, more characteristics about each game could be added as well, splicing together multiple different databases into a master list. Number of penalties, how many points were scored each quarter, and whether the game went to overtime or not are some examples that could be added. A feature that could also be beneficial to improve functionality, especially when more variables are added to the dataset, is a way to measure how useful a specific feature is when predicting the outcome. This would show irrelevant features, allowing for only useful variables to be used for both accuracy and performance optimization.

References

[1] *Pro Football Statistics and history*. Pro. (n.d.). Retrieved October 25, 2021, from https://www.pro-football-reference.com/.